Certified Tester
Game Testing (CT-GaMe)
Syllabus

Version 1.0.1

International Software Testing Qualifications Board

Provided by

Russian Software Testing Qualifications Board (RSTQB)

Copyright Notice

Revision History

Version	Date	Remarks
v1.0.1	October 21, 2022	GA Released

Table of Contents

Acknowledgments

This document was produced by the core team of the Russian Software Testing Qualifications Board (RSTQB) consisting of:

Andrey Konushin (President of the Board)
Margarita Trofimova (Vice-president of the Board)
Alexander Alexandrov (Editor-in-Chief)
Vadim Lukovaty
Maksim Nikolaev
Pavel Sharikov
Alexander Torgovkin
Svetlana Yushina

The core team thanks the review team for their suggestions and input. Russian Software Testing Qualifications Board would like to acknowledge and thank LLC "Bytex" for their contribution to the development of this syllabus.

In addition, the core team would like to acknowledge and thank the leaders and members of the Working Groups for their early and ongoing guidance: Galit Zucker (General Secretary), Graham Bath (chair, Specialist Working Group), Gary Mogyorodi (member, Glossary Working Group) and Klaus Skafte (chair, Exam Working Group).

The following persons took part in the reviewing, commenting or balloting of this syllabus or its predecessors:

Ivan Pchelkin	Natalia Sterkhova	Irina Yakovleva
Matthias Hamburg	Chunhui Li	Blair Mo
Tal Pe'er	Wim Decoutere	Claude Zhang
Meile Posthuma	Nishan Portoyan	Francisca Cano Ortiz
Wojciech Becla	Jana Gierloff	Paul Weymouth
Nitzan Goldenberg	Jürgen Beniermann	Erwin Engelsma
Georg Haupt	Gary Mogyorodi	Péter Sótér
Bíró Ádám	Darvay Tamás Béla	Laura Albert
Gergely Ágnecz	Graham Bath	Mike Smith

This document was formally released by the ISTQB® on 21.10.2022.

0. Introduction to this Syllabus

0.1 Purpose of this document

This syllabus forms the basis for the ISTQB® Certified Tester Game Testing. The ISTQB® provides this syllabus as follows:

1. To National Boards, to translate into their local language and to accredit training providers. National Boards may adapt the syllabus to their particular language needs and modify the references to adapt to their local publications.
2. To Exam Boards, to derive examination questions in their local language adapted to the learning objectives for each syllabus.
3. To training providers, to produce courseware and determine appropriate teaching methods.
4. To certification candidates, to prepare for the exam (as part of a training course or independently).
5. To the international software and systems engineering community, to advance the profession of software and systems testing, and as a basis for books and articles.

The ISTQB® may allow other entities to use this syllabus for other purposes, provided they seek and obtain prior written permission.

0.2 The Certified Tester Game Testing

The Certified Tester Game Testing qualification is aimed at anyone involved in software testing who wishes to broaden their knowledge of game testing or anyone who wishes to start a specialist career in game testing. The qualification is also aimed at anyone involved in game software engineering who wishes to gain a better understanding of game testing.

The syllabus considers the following principal aspects of game testing:

- Technical aspects,
- Method-based aspects,
- Organizational aspects.

0.3 Career Path for Testers

Building on the Foundation Level, the Game Testing supports the definition of career paths for professional testers. A person with the Game Testing certificate will have extended the broad understanding of testing acquired at the Foundation Level to enable him or her to work effectively as a professional tester in a game project.

Those possessing a Game Testing certificate may use the acronym CT-GaMe.

Please visit [ISTQB-Web] for the latest overview of ISTQB®'s career path.

0.4 Business Outcomes

This section lists the Business Outcomes expected of a candidate who has achieved the Game Testing certification.

GaMe-1	Describe basic concepts of video games and game software testing
GaMe-2	Determine risks, goals and game software requirements under the needs and expectations of stakeholders
GaMe-3	Conceptually design, implement and execute basic game software tests
GaMe-4	Know the approaches to game software testing and their purpose
GaMe-5	Recognize the tools supporting game testing

GaMe-6 Determine how testing activities align with the software development lifecycle and reduce the cost of developing and publishing video games

0.5 Examinable Learning Objectives and Cognitive Level of Knowledge

Learning Objectives support the achievement of business objectives and are also used to design examinations for the certification in Game Testing. Each learning objective refers to a cognitive process (K-level).

A K-level, or Cognitive level, is used to classify learning objectives according to the revised taxonomy from Bloom [Anderson01]. ISTQB® uses this taxonomy to design its syllabi examinations (see Appendix A for further details).

This syllabus considers three different K-levels (K1 to K3):

K-level	Keyword	Description
1	Remember	The candidate should remember or recognize a term or a concept.
2	Understand	The candidate should select an explanation for a statement related to the question topic.
3	Apply	The candidate should select the correct application of a concept or technique and apply it to a given context.

In general, all parts of this educational program contain learning objectives for level K1. In other words, a candidate for a certification must recognize, remember and reproduce a term or concept. The learning objectives for Levels K2 and K3 are indicated at the beginning of the chapters that contain them.

0.6 The Certified Tester Game Testing Certificate Exam

The Certified Tester Game Testing Certificate exam will be based on this syllabus. Answers to exam questions may require the use of material based on more than one section of this syllabus. All sections of the syllabus are examinable, except for the Introduction and Appendices. Standards and books are included as references, but their content is not examinable, beyond what is summarized in the syllabus itself from such standards and books.

Refer to [ISTQB_EXAM_S&R] for the Certified Tester Game Testing Exam Structure and Rules.

The ISTQB® Certified Tester Foundation Level certification [ISTQB_FL_SYL] shall be obtained before taking the Certified Tester Game Testing certification exam. However, it is strongly recommended that candidates also:

- have at least a minimal background in either game software development or testing, such as 1 year experience as a game software user acceptance tester.
- take a course that has been accredited to ISTQB® standards (by one of the ISTQB-recognized member boards).

0.7 Accreditation

An ISTQB® Member Board may accredit training providers whose course material follows this syllabus. Training providers should obtain accreditation guidelines from the Member Board or body that performs the accreditation. An accredited course is recognized as conforming to this syllabus, and is allowed to have an ISTQB® exam as part of the course.

The accreditation guidelines for this syllabus follow the general Accreditation Guidelines published by the ISTQB® Processes Management and Compliance Working Group.

0.8 Handling of Standards

There are standards referenced in the Certified Tester Game Testing Syllabus (e.g., (IEEE, ISO, etc.). The purpose of these references is to provide a framework (as in the references to ISO 25010 regarding quality characteristics) or to provide a source of additional information if desired by the reader. Please note that the syllabus is using the standard documents as reference. The standards documents are not intended for examination. Refer to section 8. References for more information on Standards.

0.9 Keeping It Current

The software industry changes rapidly. To deal with these changes and to provide the stakeholders with access to relevant and current information, the ISTQB® working groups have created links on the [ISTQB-Web] which refer to supporting documents, changes to standards, and new occurrences in the industry. This information is not examinable under this syllabus.

0.10 Level of Detail

The level of detail in this syllabus allows internationally consistent courses and exams. In order to achieve this goal, the syllabus consists of:

- General instructional objectives describing the intention of the Certified Tester Game Testing.
- A list of terms that students must be able to recall,
- Learning objectives for each knowledge area, describing the cognitive learning outcome to be achieved,
- A description of the key concepts, including references to sources such as accepted literature or standards.

The syllabus content is not a description of the entire knowledge area of game testing; it reflects the level of detail to be covered in Certified Tester Game Testing training courses. It focuses on test areas and techniques that can apply to most game projects.

0.11 How This Syllabus is Organized

The Certified Tester Specialist Game Testing syllabus contains seven chapters covering the knowledge necessary to be a Game Testing Specialist.

The top-level heading for each chapter specifies the minimum time for the chapter; timing is not provided below chapter level. For accredited training courses, the syllabus requires a minimum of 15 hours and 25 minutes of instruction, distributed across the seven chapters as follows:

- Chapter 1: 75 minutes Specificity of Game Testing
  - The tester learns the basic principles related to game software testing, the reasons why game testing is required, what the test objectives are, and the difference between game testing and playing the game.
  - The tester understands the game test process, the major activities, and work products.
- Chapter 2: 180 minutes Testing Game Mechanics
  - The tester learns to distinguish between different game mechanics and its testing approaches.
  - The tester learns both dynamic and static methods used for game mechanics testing.
- Chapter 3: 165 minutes Graphics Testing
  - The tester learns about features and types of the graphic content of the game product and the best practices of graphics test execution, including its tool support.
  - The tester learns about the areas of responsibility of the persons involved.
- Chapter 4: 190 minutes Sound Testing

    - The tester learns about types and technologies of the sound content of the game product, best practices of sound test execution including its tool support.
    - The tester learns about the areas of responsibility of the persons involved.
- Chapter 5: 65 minutes Game Level Testing
    - The tester learns the dependency of the level design and testing approaches on the genre of the game and its tool support.
    - The tester learns about areas of responsibility of the persons involved.
- Chapter 6: 95 minutes Game Controllers Testing
    - The tester learns about types of game controllers including related defects and their possible causes.
    - The tester learns approaches to testing controllers and its tool support.
- Chapter 7: 155 minutes Localization Testing
    - The tester learns the basics and differences of localization and internationalization, its risks, approaches to testing and its tool support.
    - The tester learns about the areas of responsibility of the persons involved.

1. Specificity of Game Testing – 75 minutes (K2)

Testing Keywords
Functional testing, playtest

Video Game Specific Keywords
3D model, concept stage, video game designer, multiplatform, post-production stage, pre-production stage, production stage, video game

Learning objectives for Chapter 1

1.1 Game Testing Basics
GaMe-1.1.1 (K1) Recognize objectives and specifics of game testing
GaMe-1.1.2 (K2) Give examples of product risks in game software
GaMe-1.1.3 (K2) Give examples of specific defects related to game testing
GaMe-1.1.4 (K2) Summarize how the risks of game testing can be mitigated
GaMe-1.1.5 (K2) Compare the activities of game testing with those of playing

1.2 Typical Roles of the Game Development Team
GaMe-1.2.1 (K1) Recognize specific roles and tasks in the game development team

1.3 Testing Activities throughout the Game Software Development Lifecycle
GaMe-1.3.1 (K1) Recall testing activities throughout the game software development lifecycle

1.1 Game Testing Basics

1.1.1 Specifics of Game Testing

In software testing, it is customary to distinguish between various specializations. Some testers are engaged in testing the performance of applications, others focus on testing mobile applications, others are looking for vulnerabilities in security systems, and some are looking for defects in computer games.

Games may vary in genre and be intended for a single or multiple players. However, approaches to testing games may be considered as independent of these characteristics. Section 2.1 of this syllabus describes in detail functional and non-functional test approaches of game mechanics.

It is important to note that the terms “game”, “computer game” and “video game” are treated as synonyms in this syllabus. This is to make the text more readable and within the context of the syllabus. A game can only be considered as software and cannot be confused with sporting games or gambling.

1.1.2 Game product risks

The explosive development of the video game industry has led to their distribution on many different platforms. Games have become larger, more complex and more demanding on technical resources. At the same time, the audience of players has increased significantly, becoming more sophisticated and demanding of game quality.

Common software project and product risks apply to video games and game development. As a result, games, like other software, are subject to testing. Some risks are specific to the gaming area of software and should be given more attention. Such risks include, but are not limited to:

- Gaining unfair advantage by cheating
- Dependence on market success from the subjective opinion of the players
- Multiplatform issues

- Complexity of predicting the effectiveness of game mechanics (see section 2.1).

Testers can find defects in the following areas:

- In the architecture of gaming software early in development lifecycle
- In the sound design
- In the text of a video game that is already available for release.

1.1.3 Defects Related to Game Testing

There are a number of defects that are especially common in video games. These include:

- problems with graphics and animation
- violation of the physical laws of the video game world.
- defects in the video game artificial intelligence
- inaccuracies in the construction of the video games plot
- incorrect functioning of some interface elements (e.g., video game characters emerging from the air, not attacking enemies, and hovering cars.)

Although such defects may be easy to spot, identifying the exact steps to reproduce them is often difficult. This complexity makes game testing challenging. A good tester should design tests in accordance with the intended game scenario, verify that it meets the specification, analyze possible deviations from the designed scenario, and report on the consequences of these deviations.

Negative testing

When testing games, just like any software, it is important to identify obvious defects and strive to find implicit defects that appear as a result of non-standard actions performed by the user. For example, instead of fighting opponents and looking for a key to a locked door, the user can collect boxes from all over the level, make a ladder out of them, and climb over the fence using them, thus avoiding a tough fight, which was not the intention of the game.

Other defects may occur if the interaction with other objects is set incorrectly, such as for a 3D model of a building used in the game. This type of defect might incorrectly allow a player-controlled character to pass through a wall, which can spoil the overall user experience and give players illegal gaming advantages. This is especially critical in multiplayer games. Hiding behind such a wall, the player will be hidden from opponents, but will be able to shoot at them.

Risks such as these raise the importance of performing negative testing for game projects. There may be a large number of players who do not want to play as the developer intended, but try to “break” or bypass the system in order to win more easily and spend less effort. To achieve this, it is not always necessary to use third-party software or special commands. It may be enough to find the "correct” defect. Such players deliberately look for defects in the logic of the game and use them for personal advantage, such as gaining access to endless resources or powerful weapons. In monetized games, this could also lead to a reduction in the profits of the game owner.

Dependence on the opinion of the players

Gaming software, like no other, depends on the subjective and often emotional judgment of end players. For software designed to solve business problems, its functionality is the most important factor. For gaming software, the most important factor is the user’s level of interest and their impressions of the gaming session. Players might be able to “close their eyes” regarding some functional defects, but if the game turns out to be boring, monotonous or uses outdated graphics, the users may simply may stop using it. A significant portion of the video game audience makes decisions about buying or using a product based primarily on feedback from game critics, reviewers and other opinion leaders.

Multiplatform

An important feature of gaming software is that it is often used on multiple platforms. In an effort to reach as large an audience as possible, game studios/publishers release it on a wide variety of platforms, including various personal computer (PC) builds, web, mobile devices and consoles.

Every update has to be tested for each available platform. This gives rise to serious risks and significantly increases the testing time required.

The game may well run well on a medium-sized computer, but have a wide variety of defects on the latest generation of consoles. Defects are also possible due to inefficient communication between various technologies or incomplete requirements for porting a game from the platform for which it was originally developed to another platform belonging to a third-party (outsourced) organization.

To mitigate these risks, it is recommended to devote more time to developing and testing the game software, testing the functionality and performance of the game on many different hardware configurations, and performing tests based on the specific features of each platform.

Testing of video games on game consoles is a particular aspect to consider. A game console is a device designed exclusively for games and does not contain any other software, like mobile devices or computers.

There are very few differences between platforms when testing video games using black-box test techniques. However, each game console manufacturer may have its own specific requirements that a video game must comply with before it can be published. These requirements are proprietary documents provided to developers and publishers under confidentiality agreements. Each checklist of requirements consists of several categories, and the game must comply with them to avoid being rejected by the console manufacturer [URL1], [URL2].

Therefore, testers of video games on consoles must test for compliance with these requirements in addition to the standard software testing methods.

It may be necessary to use special equipment to detect, identify and understand the reasons for the manifestation of defects. This equipment is essentially the same console, but provides additional modes that help in the development and testing of games. The console is registered on the site for authorized developers and testers, after which it is activated and access to the special modes is granted.

1.1.4 How testing mitigates product risks of video games

Most video games, especially open-world games, are so complex that testing every possible combination of game objects, events, and factors is impossible. Even carrying out single tests of each combination during the entire software development lifecycle could be lengthy, and confirmation testing after changes in the next version of the game requires additional effort.

Risk management is used to efficiently allocate resources and assess the possible defects based on their likelihood of occurrence and impact on the game.

For each test case, two questions are asked to determine the priority of the test:

1. How likely are players to find a defect here?
2. If they find a defect, how will it affect the players and the company that owns the game?

For example, if a game has a linear structure, and the user accesses its content step by step, then defects that appear during the first game session most probably will have a higher priority (but not always a higher severity) to be fixed than defects which occur at the fiftieth hour of play. If a loyal user has already spent a lot of time in the game, they are more likely to take the encountered defect more calmly, and it will not have

much effect on the user's overall impression of the game. It is another matter if such a defect impacts all new players during the introductory training. In this case many players may refuse the game product.

To reduce risks by testing, it is important to identify groups of defects rather than looking for individual defects in the game. This allows developers to simultaneously solve a problem in the game's code or architecture and provide an improvement in the quality of the final product.

Defect groups are clusters of defects in critical areas. There are areas in the game products that are likely to have defects, such as graphics, sound, localization, client-server interaction, and hardware [Nystrom14].

It is important to gain player loyalty to the game when they are learning to play it or taking a tutorial. At this stage, a huge number of game functions are available to the player so that they can assess the product. As a result, this early stage is usually declared as being critically important.

1.1.5 The difference between testing and "playing"

One might get the impression that the essence of the job of computer game testers is to play online with colleagues all day long. But this statement is far from the truth.

The tester rarely manages to just play, except while getting to know a new project or during playtests. Most of their work relates to conducting various types of testing, which in many respects are in common with the types of testing used for other software.

The average user starts the game to pass it, beat opponents or have a good time, whereas the tester verifies that the game meets the requirements written in the specification. Moreover, the tester's personal preferences may not map to the intended target audience of the game itself. Nevertheless, the tester will go through the same level over and over again to test, for example, that the game objects are displayed correctly. The monotony experienced by the tester in running these repeated actions is compensated by the satisfaction they get from finding defects that could disrupt the gameplay conceived by the developer.

Relation to other syllabi

The approaches discussed in this syllabus apply exclusively to game testing. They apply to both software testing (e.g., game mechanics testing) and hardware testing (e.g., controller testing).

Some aspects of testing a game are not so different from testing other software, in particular when executing test types like regression testing, performance testing or usability testing. Fundamental testing concepts, non-functional testing techniques, and features of software testing on mobile platforms are covered in detail in other ISTQB® syllabi.

1.2 Typical Roles of the Game Development Team

In most cases, a computer game development team includes (but is not limited to) roles that are not fundamentally different from those found in other areas of software development. These include business owners, project managers, analysts, developers, game designers and tests. In small teams, one person might take on several roles at the same time. The following is a summary of the main functions that can served by typical roles in a gaming software project. Additional roles that may be included in a team for special areas of game development are described in appropriate chapters of this Syllabus.

Role	Responsibilities
Business Owner	• Develops business • Defines marketing policy • Initiates new products

Role	Responsibilities
	• Can take part in team building
Project Manager	• Sets and monitors the execution of tasks • Keeps track of the stages of software development lifecycle
Analyst	• Identifies product requirements • Works with the original specification • Updates specifications, keeps them up-to-date.
Developer	• Develops and debugs code • Corrects defects • Develops component tests.
Game Designer	• Designs game mechanics, balance, plot, levels, game economics • Creates game documentation • Thinks over and develops the game content.
Tester	• Develops a test model and testing strategy • Designs and prepares test data • Reviews specifications • Performs various types of testing • Finds and analyzes defects

1.3 Testing Activities throughout the Game Software Development Lifecycle

Concept stage

The main goal of this stage is to achieve a common vision of the future product between the development team and the customer. Ideas are drawn up in the form of documents which briefly describe the general vision of the game, its genre, setting, the essence of the gameplay, mechanics and its main features.

Artifacts that might be reviewed at this stage include:

- Vision document,
- Concept document,
- Product and technical restrictions.

Testware produced at this stage include:

- Test strategy,
- Draft of test plan,
- High-level test effort estimation,
- Test environment.

Pre-production stage

The main goal of this stage is to create a prototype of a future game which is an initial working version of the product. The prototype implements the main gameplay in draft quality. At this stage, the core game mechanics and initial hypotheses are tested (see Chapter 2), and the technical capabilities of the team are

assessed. Testers, game designers, and other members of the development team can take part in testing the prototype.

The development team reviews draft requirements to establish the possibility of implementation and the test team reviews them to identify the interaction of the mechanics with each other and functionality of the assembled prototype. In addition, testers are involved in the creation of the test plan and test schedule documents.

As the documentation for the game is created, the test plan is written and updated, and the documentation is reviewed. These measures mitigate technical risks of the product at the pre-production stage.

Artifacts that might be reviewed at this stage include:

- Requirement specifications,
- Game prototype,
- Test artifacts created at the previous stage.

Testware produced and updated at this stage include:

- Test strategy,
- Test plan,
- Test schedule,
- Test environment.

Production stage

During the production stage, the game product is created. This is the longest stage and is usually broken down into separate parts for the Alpha and Beta versions of the game.

At this stage, testers finalize checklists and sets of test conditions, and execute the functional and non-functional testing of implemented product components. Component integration, system and acceptance testing are performed (see [ISTQB_FL_SYL]). This stage is where the largest number of defects is identified, which results in testers conducting confirmation testing and regression testing.

Testware produced or updated at this stage includes:

- Test plan,
- Test schedule,
- Test cases, test conditions, checklists, test suites, scripts,
- Defect reports,
- Test reports.

Post-production stage

The post-production stage for game testing includes common testing activities during the software maintenance period. This typically involves testing of updates and regression testing, as described in [ISTQB_FL_SYL].

2. Testing Game Mechanics - 180 minutes (K3)

Testing Keywords
Ad hoc testing, functional testing

Video Game Specific Keywords
Client mechanics, concept stage, core mechanics, video game designer, video game mechanics, meta mechanics, pre-production stage, production stage, server mechanics, video game state

Learning objectives for Chapter 2

2.1 Game Mechanics
GaMe-2.1.1 (K2) Classify the types of game mechanics
GaMe-2.1.2 (K2) Differentiate the testing of gameplay mechanics and non-gameplay mechanics
GaMe-2.1.3 (K2) Differentiate the testing of core mechanics and meta mechanics
GaMe-2.1.4 (K2) Differentiate the testing of client, server, and client-server mechanics
GaMe-2.1.5 (K2) Give examples of defects in game mechanics

2.2 Approaches to Testing Game Mechanics
GaMe-2.2.1 (K2) Summarize main approaches and test objects at different stages of creating a game product
GaMe-2.2.2 (K2) Distinguish the importance of testing game mechanics
GaMe-2.2.3 (K2) Distinguish the importance of reviewing documentation describing game mechanics
GaMe-2.2.4 (K3) Apply the fundamental approaches of testing game mechanics

2.1 Game Mechanics

2.1.1 Types of Game Mechanics

Chapter 1 mentioned that for games, one of the most important things is to create user interest in order to retain and increase the audience. Players evaluate the graphics of the game, the gameplay itself, and the mechanics that make up the gameplay.

Video game mechanics are at the heart of any game. Video game mechanics are a method of interaction between the game and the user. It considers the impact and feedback between game and user and the changes in the video game state within the specified requirements. Mechanics can be divided into different types, depending on different aspects and goals of the game.

Depending on the type of player interaction with the mechanics:

- Gameplay mechanics relate to when the user consciously interacts with the gaming system, basing their actions on the availability of information. They can influence the change in the video game state, while clearly seeing the result of their actions.
- Non-gameplay mechanics relate to when the user cannot or can only partially influence the state of the gaming environment through their actions. They do not recognize their influence, because part of the game mechanics is hidden from them.

Depending on the impact on the main gameplay:

- Core mechanics form the basis of the game and define actions that the user repeats throughout the game. These are aimed at ensuring that the user receives a certain experience which was embedded into the game by its creators.
- Meta mechanics are outside the main gameplay but can influence it. They are aimed at making the player do what the game designer wants (e.g., to return to the game, keep playing, or make

purchases). These mechanics can be combined with core mechanics to make the game more interesting.

Depending on the architecture of the structure of the game:

- Client mechanics relate to processing of the user's actions occurs exclusively on the user's device.
- Server mechanics relate to the mechanics which are processed only on the game server.
- Client-server mechanics relate to the mechanics which are processed both on the user's device and on the server. In this case, there is a constant exchange of data between the server and the client.

2.1.2 Difference between Testing Gameplay Mechanics and Testing Non-Gameplay Mechanics

The user does not interact directly with all the mechanics implemented in the game.

Gameplay mechanics may be described in a formal or informal specification, but non-gameplay mechanics must be described to the user. Often, they directly affect the achievement of the goal of the gameplay. For example, picking coins by a game character must be performed within a limited time. The functionality of the movement of picking the coins and increasing the number of coins is an example of the game mechanics.

Non-gameplay mechanics are usually hidden from the user, although they can be triggered by the action of the user themselves. For example, when the user makes a purchase of any product within a game, a sequential flow of non-game mechanics is started. For example:

1. A request is made to the database to get the state of the user's game account
2. A transaction is made to withdraw funds
3. The purchase is logged, i.e., information about the performed action is recorded in a special text file
4. An email with the details of the purchase is sent to the user

Additionally, the date of purchase and the type of purchased product can be recorded, so that an automatic mailing about a discount for a similar product will be sent. Non-gameplay mechanics must be specified as formal requirements as they are not evident.

Testing of gameplay and non-gameplay mechanics is functional testing, as described in [ISTQB_FL_SYL].

2.1.3 Difference between Testing Core Mechanics and Meta Mechanics

The difference in approaches for testing core and meta mechanics depends on their impact.

Removing a core mechanic from the game completely changes the gameplay. For example, if the car movement is removed from a racing simulator, the game can no longer be classified as “racing”.

Examples of meta-mechanics would be choosing a car model with specific parameters depending on the type of race track, or buying and equipping a player character with new armor before entering a battle. Removing a meta mechanic from the game rarely changes the essence of the entire game.

In addition to functional requirements, core and meta-mechanics may have other non-functional requirements which directly affect how they should be tested.

For example, for a meta-mechanic to change the appearance of a playable character, the most important thing is that it functions correctly. Even if this mechanic works with a slight delay, such that the appearance changes a couple of seconds after the player presses the button, neither the tester nor the user will consider this a defect. However, for the core mechanics of a character jumping or using ability in battle, such a delay

would be critical, because it could prevent the player from winning. As a result, performance testing is also important for core mechanics.

The essence of core and meta mechanics also affects when they are tested. As a rule, their verification usually begins at different stages of game development. Core mechanics, such as the movement of a game character, receiving a reward, etc., appear already in early versions of the product. At the same time, verification of functional correctness and the interest level of the target audience begins.

Meta mechanics are usually added to the game and begin testing later, during the production stage. The assessment of their impact on the average length of a game session, the number of invited friends, the size of the average payment, the most popular game products, etc., takes place at the beta testing stage. At the same time, A/B testing is often used to determine the most effective mechanics.

2.1.4 Difference between Testing Client, Server and Client-Server Mechanics

The need to test client side, server side and client-server mechanics depends on the architecture of the game itself. Each of these types of mechanics requires a different approach to testing depending on the specifics of their implementation.

Client mechanics are processed on the devices of the players themselves, so the user can access all the data stored on the client. Some games can be completely run on the client and do not use the server at all, i.e. all mechanics are client side. These games are usually single-player and do not require an internet connection.

The game client contains the code that is executed on the player's device and all the parameters, damage calculation formulas, game levels, character models, etc. As a result, the player, if desired, can change all the data in the game client. By modifying the client, the user can, for example, increase the speed of his technique, the amount of character health, the value of the reward for completing the task, etc.

Such client changes may give the player an in-game advantage, but in single-player games this is generally not considered a major defect. For example, in a single-player game, the mechanics of sword fighting with computer opponents might be implemented. In this case, the parameters of the weapon will not be validated on the server. Therefore, if a specific player makes a change in the game client and increases the damage of his sword, this will only change his personal gameplay. It will not affect the interests of other independent players who play the same game on their devices.

Client mechanics can also be present in multiplayer games, but they still won't affect other players. For example, the mechanics of viewing a level map, opening a character's inventory, etc. are processed only on the client side and verified by functional tests without server involvement.

Testing of the client mechanics is usually carried out using black-box testing. When doing this, the tester may need to check the user interface of the application. In this case the user interface itself can be used when testing the game to get actual results.

Server mechanics are implemented exclusively on the server side. Due to this feature, the logic of the mechanic is protected from user intervention. Players cannot directly influence the processes that run on the remote server, so the developers transfer the most important parts of the game logic calculations there.

Testing server mechanics is most important for multiplayer online games. For example, when a player starts a team competition in player versus player (PvP) mode, a selection of allies and opponents is conducted on the server with approximately equal game ratings and character levels. Otherwise, the competition might turn out to be uninteresting and uncomfortable for all participants.

The approach to testing functionality implemented on the server is different from testing client mechanics. Here, the tester generally does not need a user interface. Testing takes place in server consoles or using special tools. The tester spends most of their time analyzing logs or interacting with the database.

For server mechanics, the most important testing types are functional testing, performance testing and security testing.

Client-server mechanics involve both client and server sides and combine the features of the two previously mentioned types of mechanics. One part of the logic is processed on the players' devices, the other on the remote server. Parts of such mechanics constantly "communicate" with each other, sending data among themselves. In a multiplayer game, the server receives data from the client, and then sends the result of the actions to all the server players who are affected by these actions.

The data arriving at the server undergoes mandatory validation. For example, if a user wants to buy something in the in-game store, then the server must check that there is a sufficient amount of in-game currency available for a purchase, depending on the progress of the player. Even if the player tries to modify the client part on his device and send false parameters to the server in order to gain a gaming advantage, the server will replace them with correct data.

When testers test client-server mechanics, they create one or more test cases in such a way as to cover the maximum number of different mechanics.

An example of such a test would be a client logging into a game server and killing a computer-controlled opponent. Despite the small number of necessary actions that make up this test, almost all the mechanics of client-server interaction are used here:

- Sending a request from the client to the server responsible for authorization,
- Making a request from the authorization server to the database, performing the authorization, and restoring the previous video game state,
- Selecting maps already loaded on the server or creating new game mechanics on the serve,
- Connecting the client to the game mechanics server where the map is located,
- Providing a view of the game character and game objects in the character's vicinity while the server creates objects and gives the client the command to display those to the user,
- Moving the character around the map while displaying new game objects,
- Finding a computer enemy and its attack and displaying the attack on the client
- Receiving the “attack” command from the server,
- Verifying that an attack is possible by ensuring the character and the mobile object (Mob) are at an acceptable distance for an attack, and that there are no objects between the character and the Mob preventing the attack from being carried out,
- When an attack is possible, sending a message by the server to the Mob about receiving damage
- Calculating the amount of damage received by the Mob on the server, reducing the value of the parameter responsible for the health of the Mob,
- Displaying the damage done to the player by the Mob.

2.1.5 Examples of Defects in Game Mechanics and Possible Causes of Their Occurrence

Game mechanics involve influences on game objects and feedback signaling the result of these influences. Taken together, this creates the unique character of the game, with suitable game dynamics. Most often, games use a wide range of influences and feedback elements.

The following types of defects are associated with game mechanics:

- Functional defects in mechanics,
- Defects of appropriateness recognizability of the game,
- Defects of effectiveness of mechanics manifested in a specific game environment.

Functional defects of the game are mentioned the most when players talk about defects in game mechanics. For example, weapons do not reload, loot is not collected, the image does not get closer when

using binoculars. Such defects usually arise from errors made by the developer in the game code. They are quite noticeable and are quite easy to detect and fix.

Defects of appropriateness recognizability of the game are related to getting responses from the game while using a particular mechanic. The situation is incorrect when the player does not understand why the game is over. Therefore, game mechanics are often accompanied by visual and sound effects. This can be the visual effect of an explosion, the sound of a time counter, or even a text message. The very presence of a response when using mechanics is tested and the justification of its presence or absence is assessed. The operation and correctness of the effect is tested by other types of testing (graphics or sound testing).

The third type of defect is related to the effectiveness and scope of mechanics. The mechanics can be effective on their own, but might stop working within the gameplay. To avoid such a situation, mechanics are tested in conjunction with other mechanics, objects, and other necessary content at the game level. Integration testing is therefore conducted together with usability testing to check the effectiveness of the mechanics in the gameplay.

For example, a game designer might add two opponents to the game with different behaviors and characteristics. The first is fast and jumpy, but deals a small amount of damage, and the second is slow, clumsy, but strikes accurately and strongly. In terms of characteristics, both opponents look equal. They have their own advantages and disadvantages. Each can have his own algorithm of behavior, and in theory they should pose about the same amount of danger to the player. However, if a large part of the game level comprises various hills that the player character can climb, the situation changes. Being at a height which is inaccessible to a slow enemy, the player can safely attack the opponent. In this case, the user will have difficulty confronting another enemy, who can also use hills, but moves at high speed and jumps high. The solution to this problem can be both a change in the mechanics itself, and a change in the location of objects on the level.

The complexity of such tests is associated primarily with the difficulty of predicting such situations and how the mechanics will work in them. Therefore, the search for problems of interaction between mechanics and objects of the environment is often carried out as part of ad hoc testing at the beta testing stage with a large group of players.

2.2 Approaches to Testing Game Mechanics

2.2.1 Procedures and Approaches for Testing Game Mechanics throughout the Software Development Lifecycle of a Game Product

At different stages of software development, testing of mechanics in games can be carried out using various testing approaches.

At the stage of creating a game prototype, only core mechanics are implemented. Here the tester's main tasks are to test game mechanics against the following characteristics:

- Functional correctness,
- The level of interest for the players,
- The attractiveness for the players.

All mechanics and their rules of operation are reviewed to ensure they are fully and unambiguously described in the video game design document. All allowed options for the interaction of the user with the mechanics should be worked out, and the impact of each mechanic on the entire gameplay should be determined.

During the production stage, the implemented mechanics are subjected to functional testing to verify that they meet the stated requirements. As a rule, testing here takes place using sets of test conditions and checklists. The sequence of tests should aim for the maximum coverage of all aspects of the mechanic.

Exploratory testing and ad hoc testing are more effective for testing the interaction of different mechanics with each other. This is especially evident in large-scale multiplayer games, where the number of ways of interaction between mechanics and various components of the game becomes too great to be reflected in test conditions.

As part of non-functional testing, the effect of mechanics on the overall performance of the game is tested. Resource utilization (e.g., RAM, CPU, etc.) and actual result (e.g., occurrence of freezes in the game client, significant decrease in the frame rate, etc.) are tested. The compatibility of various components and software is to be tested, especially with antivirus programs.

If the game requires the presence of a server, the server performance is also tested. Performance testing is planned and executed during the production stage, when planned mechanics that could affect performance are being implemented. Details of planning and execution of performance testing are described in [ISTQB_FL_PT].

Considerable attention is paid to meta mechanics when conducting beta testing with a large number of players. The convenience and clarity of the game is assessed for players. As with testing non-gaming software, the main purpose of this testing is not to find obvious functional defects, which ideally should be identified in the previous stages, but to get feedback from end players. However, end players can also participate and provide feedback in earlier alpha tests in, for example, large multiplayer games.

After the game is released to the market, the task of the testers is to handle and check for defects in the mechanics reported by the players, and to test any new mechanics which are added to the game.

2.2.2 The Importance of Testing Game Mechanics

It is the game mechanics that are the backbone of the game. It forms the basis for the player's experience, which is necessary in order to keep them interested. Since game mechanics are the essence of any game, testing their correct operation is of the highest priority for a tester, and defects found during such testing are usually the most critical.

A defect in the core mechanics, such as a character's inoperative jump, can lead to the inability to overcome a certain obstacle and, as a result, to complete the game.

But even if the mechanics are rarely encountered in the game or are associated with activities that are unnecessary for passing the video game, (e.g., the ability of the character to ride a horse), its incorrect operation will negatively affect the perception of the game as a whole.

2.2.3 The Importance of Review of Game Mechanics

An important stage in the creation of a game is the development of game documentation, which, among other things, describes the game mechanics.

Reviewing the specification of game mechanics early in the development stage helps to avoid many problems later on. These problems may be associated with the development process itself, as well as with the reaction of players to the implemented game mechanics in the final product.

During the review, the tester should pay attention to the following quality characteristics [ISO 25010]:

- Completeness of the description,
- Compliance with reality,
- Structure and ease of navigation in the documentation.

Problems in the game development process that a review can prevent:

- Incorrect understanding of the essence of mechanics by the developer and the incorrect evaluation of effort required, due to an insufficiently clear, detailed and unambiguous description of the mechanics,
- The incompatibility of new mechanics with existing ones,
- The inability to implement mechanics in a particular project.

Problems related to player perception of mechanics that a review can prevent:

- General inappropriateness of mechanics in a particular game.
- Uninteresting mechanics,
- Unbalanced mechanics,
- Too rare use of mechanics in the game.

2.2.4 Testing the Video Game State after the Resumption of the Session and When the User Is Inactive

Video Game State

The video game state is understood as the value of all parameters and variables that describe all objects in the game at a particular moment in time.

The more complex the game is and the more possibilities for actions it provides to the player, the more parameters each object has.

All parameters can be conditionally divided into visible and hidden.

Information about the values of the visible parameters is explicitly available to the player. This can include:

- The user's level of in-game experience for the completed task,
- The characteristics of the selected weapon,
- The number of allowed items in the inventory,
- The cost of the goods in the in-game store.

Hidden parameters can be used by developers to influence the course of the game without involving the awareness of the player. For example, in addition to the Attack and Defense indicators known to the player, the character may have an accuracy parameter, which is hidden from the player and determines the probability of a shot hitting the target. Often the player is unaware of the existence of such parameters and what exactly they affect.

Hidden parameters can be responsible for different aspects of the game:

- Drop rate, i.e., the chance of obtaining a random item as a reward,
- The rate of fire or the speed of movement of equipment on various types of surfaces.

The influence of the player's choice on the development of the game's plot. A set of hidden and explicit parameters uniquely defines the game character controlled by the player and other objects.

Objectives of testing the Video Game State

An important point when testing game parameters is to test if the game really uses the actual values of the parameters.

In the case of explicit parameters, testing is performed by visually comparing the displayed value with the one actually used. For example, if a game character with an attack parameter equal to 1 inflicted 10 units

of damage on the enemy, then after the player increased the attack parameter to 2, the amount of damage done to the same enemy should also increase and become, say, 20 units.

Such a test is especially important if information about the new amount of damage is available to the player, in the form, for example, of a game hint: such as “Each unit of an attack parameter adds 10 damage units”. In this case, the player will also be able to compare the value indicated in the hint with the actual damage.

A regular player usually focuses only on the adequacy and consistency of the damage change, without considering specific values of the parameters. Such a player may not consider it as a defect to inflict the wrong number of points of damage, but if the damage does not change or decreases, this will certainly be noticed.

A professional tester is guided by the formulas and principles for calculating the change in parameters specified in the documentation. If this information cannot be obtained from the documentation, the tester, like the player, evaluates it relatively, based on common sense.

To test hidden parameters, the tester may occasionally need to access the game database. With its help, the tester can compare the actual values of the parameters with the expected ones.

In addition, if it is assumed that the user can change the explicit parameters of their character during the game, then the very possibility of changing the parameters and the correctness of their display is tested.

Testing the video game state is closely related to saving and loading a game.

Saving and Loading

Many games are impossible to complete in one session, or they may in principle involve an endless gameplay. Games therefore often use “save and load” video game state functions so that the player does not have to start from the beginning every time.

Other reasons for using saves are to encourage the user to explore the game world, find different ways to solve the problems offered by the game, and even manage the complexity of the game. If the player has the opportunity to survive in front of a difficult obstacle, the cost of their error will be lower than if a wrong action completely nullifies the progress.

Saving refers to information about the progress of the game which is stored on the hard drive or in the cloud. The saving can be presented as a file that contains information about the current state of the game. The size of such a file is several times smaller than the game itself and usually contains the following information:

- A list of objects for which state information needs to be stored,
- A list of changing parameters for each object,
- A specific code for each parameter, by which the game can determine what is happening to it at the moment.

Almost any variable can act as an object. Its state is operated upon by the game system and it needs to be remembered. Variable objects include, for example, the health level of the game character, the number of learned abilities, and the coordinates of the character on the game map.

Saving types and testing areas

The save method, the save file format, and the list of stored parameters may vary from game to game. Throughout the history of video games, a large number of approaches have been invented to save progress. The most commonly used are described below.

Using Checkpoints for saving

Saving occurs automatically at moments specified by the developers. Reaching each next checkpoint overwrites the information about the player's current progress. The video game state saved in checkpoints is stored only for the current game session and is deleted when the game ends.

Sometimes the game does not explicitly show that the user has reached the checkpoint, and the progress is saved. In order to test the correctness of saving and loading, the tester needs documentation which provides a list and exact location of all existing checkpoints. When using other methods of saving, this information in the documentation may be absent or stated in less detail.

Using stationary saves

In certain places of the game, the developers place special points at which the player, if desired, can manually save their current progress. These points are usually surrounded by a few video game objects, which avoids the creation of large save files.

The area of verification in this method is the ability to save at each point, and the ability to repeatedly save and load each saved state.

Autosaving

This is a combination of checkpoints and stationary saves. The player's progress is saved automatically throughout the gameplay at certain points. For example, many games have an autosave feature on exit. When starting a game, the user can start a new game or load a previously saved one.

The task of the tester in this case is to check that saving has occurred and the possibility of loading the achieved video game state at all points.

Manual or free saving

The user saves at any time during the game using a special item in the game menu. Fast saving may be implemented when saving and loading occurs by pressing one key or making one click.

As with autosaving, manual saving creates a special file that contains information about the video game state. Testing in this case includes checking the correctness of loading information from the file, the correct name of the file, the correct name of the file, file location in the operating system structure, correct operation in the new version of the game, and even the possibility of using the save file, for example, on another computer.

It is sufficient to test the correct loading of the required game level for games that fit the following description:

- The game assumes the only correct path and way of passing (e.g., only one correct way out of a labyrinth),
- The game character controlled by the player does not change from level to level (e.g., it does not increase the impact force, jump range, movement speed through a level, and other characteristics that could increase the speed of passing a level).

In games that fit the following description, saves contain more unique information about the game session, level, character, etc.:

- The game character may have many parameters (e.g., movement speed, range or height of the jump, impact force, inventory, appearance of the 3D model and other characteristics)
- The parameters described above can change depending on the player's in-game actions and decisions (e.g., the player finds the artifact in a labyrinth and the movement speed is increased).

For such games testers need to test the correctness of loading the required level of the game and the unique information, but also the unique information for each individual player and each individual game character. When testing the save, the tester will need to test the following:

- The player is at the game level at which he finished the previous game session,
- The playable character has an artifact,
- The speed parameter of the game character is indeed increased due to the artifact.

3. Graphics Testing - 165 minutes (K3)

Testing Keywords

Playtest

Video Game Specific Keywords

3D model, animation, collisions, video game level, hit box, level of detail (LoD), mapping, rigging, scene lighting, skinning, textures, visual effect (VFX)

Learning objectives for Chapter 3

3.1 Principles and Concepts of Game Graphics
GaMe-3.1.1 (K2) Explain features of the graphic content of a game product
GaMe-3.1.2 (K2) Classify the types of defects in graphic content

3.2 Approaches to Testing Graphics in Game Products
GaMe-3.2.1 (K2) Summarize the main approaches to artistic testing
GaMe-3.2.2 (K2) Summarize the main approaches to technical testing
GaMe-3.2.3 (K2) Summarize main approaches to gameplay testing

3.3 Graphics Test Execution
GaMe-3.3.1 (K3) Apply the fundamental approaches of graphics testing
GaMe-3.3.2 (K2) Explain the importance of testing graphics for historical validity

3.4 Tools Support for Graphics Testing
GaMe-3.4.1 (K2) Summarize the usage of graphics testing tools

3.1 Principles and Concepts of Game Graphics

3.1.1 Features of the Graphic Content of the Game Product

Any game product, either a mobile game or a large high-budget game, has a large number of different graphic elements which together allow the user to visualize the process of the game.

Different specialists perform certain functions in the process of creating graphic objects for the games:

Role	Responsibilities
Artist:	Creates various graphic content for the game.
3D modeler	Creates 3D models of the game objects
Texture artist	Creates textures for game objects
Animation specialist	Works on animation of game objects
Technical artist	Performs technical testing
Tester	tests graphics objects in the video game engine.

A tester of a game product, regardless of the budget of the game product and the amount of graphic content in it, needs to keep in mind and test its various areas that are partially, completely or separately present in each game.

Consider the main areas of the gaming product for testing.

Levels (game maps)

Levels are a separate area of the virtual world of the game, which usually represents a specific location, for example, a building or a city. The term came from early tabletop role-playing games (RPG), where it referred to the levels of a dungeon (i.e., the environment in which most of the game took place). Players started in the depths of the dungeon (level 1) and had to get to the surface going through all the levels , which became increasingly more difficult until they reached the last one (e.g., level 100), thereby winning the game. Now levels in one form or another are present in almost every video game.

Models

A model is any object in computer graphics. According to the methods of creating images, graphics can be divided into the following categories:

- 2D graphics,
- 3D graphics,
- Computer generated images (CGI) graphics.

2D graphics

2D computer graphics are classified according to the type of presentation of graphic information and image processing algorithms that follow from it. Usually, 2D computer graphics are divided into vectors and rasters, although there is also a fractal type of image representation.

3D graphics

3D graphics operate with objects in 3D space. 3D computer graphics are widely used in movies and computer games.

3D graphics can be:

- Polygonal,
- Voxels.

Polygonal graphics are a collection of vertices, edges and faces that define the shape of a polyhedral object in 3D computer graphics and volumetric modeling. The faces are usually triangles, quadrilaterals, or other simple convex polygons, as this simplifies rendering.

A voxel is a virtual element corresponding to a set of six rectangular polygons. Everything in the virtual world (i.e., pixels, polygons and voxels) must be projected onto the pixels of the physical screen. Voxel graphics are similar to raster graphics. The object consists of a set of 3D shapes, most often cubes.

Computer generated images (CGI) Graphics

CGI are 3D images obtained by a computer on the basis of calculation and used in visual arts, printing, cinematic special effects, on television and in simulators. Moving images are created by computer animation, which is a narrower area of CGI graphics.

Any image on the monitor, due to its flatness, becomes a raster, since the monitor is a matrix consisting of columns and rows. 3D graphics exist only in our imagination, since what a human sees on the monitor is a projection of a 3D figure, and the viewer creates the space by themselves. Thus, visualization of graphics can only be raster and vector, and the rendering method is only a raster (i.e., a set of pixels). The way of defining the image depends on the number of these pixels.

Textures

Texture is a bitmap applied to the surface of a polygonal model to give it color or relief illusion. Broadly, textures can be thought of as a pattern on the surface of a sculptural image. Using textures makes it possible to reproduce small surface objects that would be excessively resource-intensive to create with polygons. For example, scars on the skin, folds in clothing, small stones and other objects on the surface of walls and soil.

The quality of a textured surface is determined by texels, which represent the number of pixels per minimum texture unit.

The main aspect in texture testing is to test for the presence of textures on visible objects, for correct display, and for their uniformity. If the game textures are in high resolution, there should not be low quality textures.

Collisions

Collision is an attribute responsible for passing colliding objects through each other.

Computer games, especially console games, must distribute many of their tasks between limited hardware resources and very limited gameplay time. Despite these limitations and the use of relatively primitive and imprecise collision detection algorithms, game developers have been able to create visually believable and relatively realistic ways of displaying the interactions of game objects with each other.

In most computer games, the main objects which need to avoid collisions and intrusions are the landscape and environment of the level. These are the static, non-interactive and indestructible structures such as mountains, trees, buildings and fences. In this case, the character is represented by only one point, and the method of binary partitioning of space is used (i.e., the method of recursive partitioning of Euclidean space into convex sets and hyperplanes). As a result, the objects are presented in the form of a data structure used to efficiently perform operations on 3D computer graphics, including the sorting of visual objects in order of distance from the observer and collision detection. This provides a viable, simple and effective way to test if a point representing a character is in the environment (terrain) or not. Collisions between characters and other dynamic objects are considered and handled separately. More information on collisions can be found in Section 3.1.2.

Animations

Before the model is animated, a “skeleton” is created in it. This process is called rigging. Models of living creatures and all objects that are supposed to be animated can have virtual bones. For example, the cloak of the protagonist. The bones of the model are dependent on each other such that, for example, when the hand is displaced, the bones of the palm will also move. All subsequent animation depends on how well the rigging and skinning are performed.

Animation is a technique for creating the illusion of moving images, i.e., movement and / or changing the shape of objects. Morphing uses a sequence of still images (frames) which replace each other at a high frequency (from 12 frames per second for hand-drawn animation to 30 frames per second for computer animation).

Effects

Visual effects can be divided into two main types of tasks:

- Gameplay effects (interaction effects),
- Natural effects (environmental effects).

The principle applied depends on the specific project.

Among other genres, gameplay effects are more important in fighting games and RPGs. For example, when your playable character pushes another player's playable character, the correct interaction animation

should be played for each of them. There are other game genres, such as shooters, and especially realistic ones, such as natural effects which are just as important as gameplay. Examples of natural effects are waterfalls, fog and rain.

Scene Lighting

Scene lighting is needed for the player to view the scene. Light affects emotions. The connection between the picture and the emotional response provides another powerful tool that helps to work with character, narrative, sound and game mechanics. In this case, the interaction of light with the surface allows it to influence the brightness, color, contrast, shadows and other effects.

Due to structural features, the human eye recognizes objects in three dimensions within 110°, and full-color in an even smaller range. Since central vision (within the range mentioned above) is the first thing a human uses, it should get the critical elements that the player should definitely see as intended by the designer. Peripheral vision provides context and reinforces central vision. In a video game, if the elements that fall into the peripheral vision do not provide the necessary context or contradict the elements that are in the central vision, then the connection between the designer and the player breaks.

The following provides examples of the correct use of scene lighting:

- Light falling into the central field of view can guide the player,
- Lighting can change frames. For example, using a flashlight makes it the main light source for the player. The altered perspective captures the player's view of the illuminated area and cuts off everything else due to the strong contrast,
- Lighting can guide and also create an atmosphere, such as a sense of fear. The player is held in constant tension when, for example, somewhere in the darkness there is a terrible enemy hiding,
- The direction of the light can either make it easier to find elements at the game level or make it more complicated,
- A lack or overabundance of lighting can push the player to use special items or actions.

Scene lighting is one of the most important areas of validation because lighting creates the aesthetics of the game and influences the gaming experience. Thus, many of the scene lighting techniques seen in visual arts, cinema and architecture are used in computer games to complement the aesthetics of the virtual space and improve the player's experience. However, games are very different from movies or theater; their environments are dynamic and unpredictable. In addition to static scene lighting, dynamic light sources are used. They add interactivity and the right emotions.

Historical accuracy

Any item, such as a graphic element, animation or effect, must match the overall play style. A text or sound description of an object, model or location must also correspond to the description of the historical prototype.

Rich and complex graphic content is one of the main differences between computer games and any other type of software. This significantly changes the approach to testing. To test graphics in games, the tester must have additional knowledge of physics and optics, have an understanding of color rendering techniques, and knowledge of history. Graphics testing is one of the most critical testing activities, since it is here that the bulk of defects are concentrated, which directly affect the user's perception of the game and their gaming experience.

3.1.2 Types of Graphic Content Defects

A common category of defects in graphics testing is visual defects. Graphics defects include tearing of the image on the screen, the absence of textures, and the unexpected cropping of certain areas of the image.

When creating graphics and animation for mobile games, the same engines are used as for a PC. The difference is that same engines are adapted for specific platforms, the hardware that is used in them, and their technical capabilities [Gregory18]. Therefore, the defects that occur in such games are similar.

Lack of texture

Among the defects most frequently encountered when testing textures are:

- Lack of graphic objects' textures,
- Loss of textures during gameplay,
- Temporary textures and stubs.

As experience shows, the first two groups of defects are most often related to insufficient performance of graphics processors or outdated graphics card drivers. In comparison, temporary textures may appear due to a development defect.

Level of Detail (LoD)

In a modern 3D game, many objects located at different distances from the camera (from the player's point of view) can be displayed on the stage at the same time. In order to reduce the load on the system, developers use level of detail (LoD) technology.

When developers create a new game object (e.g., a tree, a building, or a car.), they add several variants of object models to the game. These include models with a reduced number of polygons and simplified geometry (low-pole models) and more detailed models with a large number of polygons (high-poly models). Depending on the distance to the camera, models with a different number of polygons are used to display the same object in the game. In close proximity to the camera, when it is required to achieve maximum image quality, high-poly models are used. As the camera moves away, they are replaced with less detailed models with fewer polygons. At a sufficiently long distance, the model is displayed only as a silhouette or is not rendered at all. This allows reducing the number of processed polygons and increasing the performance of the game.

For example, for a forest on the horizon, low-resolution textures can be used with only theit color is displayed, without relief and reflections. If the player gets closer, the trees will appear in full detail.

LoD technology is also used for animations, skeletons and artificial intelligence of bots, where an automated program plays a given game on behalf of a human player. For example, in shooters with a large number of enemies in the frame, bots will have different levels of detail. Bots in the background will be depicted in a simplified manner with low texture detail, whereas nearby opponents will be carefully drawn with elaborate animations and appear smart enough to fight the player.

Developers also change the level of detail depending on the current frame rate, the speed of movement of the object on the screen and the total number of simultaneously visible objects.

The defects arising from the use of LoD are mainly related to the low performance of the system. The developers try to ensure that the increase in the detail of the object or the substitution of the model occurs smoothly and imperceptibly for the player, although this is not always the case. In games running on weak hardware, low-poly models may not be replaced by high-poly models quickly enough. As a result, the player approaching an object will first see an “ugly” model, and then it will change to a better one.

Collisions

Collision is the way in which an object in the game gains size and reacts to collisions with other objects. A collision mesh or collider is created for the finished object model to interact with its environment and other objects. This an invisible simplified form of an object which is tied to it to calculate collisions with other

objects. The collider can be called the physical model of the object. It remains invisible to the player, but any collider collisions are handled by the video game engine [Buttfield19].

The shape of the collider should roughly match the mesh of the model, but often a fairly rough approximation is sufficient. For most static objects in the environment such as buildings, rocks and crashed cars, developers use low-poly colliders. This is indistinguishable in gameplay, but improves the processing efficiency of all visible objects.

In this case, the size of the collider must correspond to the visible model, especially if the player can interact directly with this object.

If the collider of the object is much smaller than its visual model, then the character will be able to “fall into the texture” or go through it. Strictly speaking, the phrase “fall into textures” (e.g., when a character can walk through another object) is incorrect, but it perfectly describes what the player sees when his character is partially or completely immersed in another object.

Such failures are most critical in multiplayer games, as it can give the player an illegal gaming advantage. For example, a tank driven into a stone will be virtually invisible to other players. At the same time, such a stone will not protect from enemy fire. This is why testers must pay additional attention to such objects on maps in player versus player (PvP) mode.

Nevertheless, some objects in the game may well be decorative and have no collisions at all. For example, a tank can easily drive through a bush, and the player does not need to be forced to go around or jump over every small pebble or tin can.

Game conventions are applied here. In the same way, one can send a character with a long weapon into a narrow low corridor to fend off enemies attacking from different directions. Unlike real life, the character will be able to turn freely in any desired direction, and will not even notice that, when turning, the spear passes through the walls. Additionally, the character's long hair may go right through the massive shoulder pads of the armor. However, the presence of a low fence, which suddenly prevents the player from climbing, will annoy the player and reduce his enjoyment.

If the size of the object’s collider is larger than the visible model, a situation may arise when the character rests against an invisible wall or stands in the air on a tiny support. Such failures are often used for unplanned fast passing of games. So-called speedrunners are specifically looking for places in locations where the developers were lazy or forgot to resize the colliders of the surrounding objects. Sometimes this allows a player to bypass a significant part of the location and reduce the passage time.

Hit boxes

If it is assumed that a character or game object can take damage, for example, from an enemy bullet / projectile, then during the battle it is necessary to calculate the position of each point on the surface of this object and determine whether there was a hit or not.

For objects of complex shape, this is very expensive and not always justified.

To simplify such calculations, the concept of a hitbox is used. For example, in 2D platformers or fighting games it can be in the form of one or more rectangles. Their position relative to each other is easy to calculate.

For a 3D object, the simplest hitbox shape is a sphere. In this case, it is sufficient to know its center and radius for calculations. In this way, at any given time, it is easy to determine whether a bullet or other object is inside this sphere. However, a parallel piped hitbox is found to be a more suitable shape because it contains less empty space compared to a sphere, and it is sufficient to know the center point of the object and three dimensions: length, width and height for calculation.

For more authenticity, the object can be described by multiple hitboxes, separately for each part of the model: body, (e.g., arms, legs, head, tail, and even weapons).

However, the parts of the model that can be damaged and that can do damage do not always coincide. Therefore, in some games, the hitbox and hurtbox are separately allocated for the same object such that areas that inflict damage (e.g., weapons, fists, etc.) and areas over which damage is inflicted (e.g., head, limbs) are defined. To determine if a character should take damage, it is tested whether the hitbox of the weapon and the hurtbox of the character's body overlap.

The main task of the tester when working with hitboxes will be to test whether the visible area of the impact matches the actual result of the attack.

Even in large games, there are moments when the player clearly sees that his weapon touched the enemy, but did not cause damage. On the contrary, if the hitbox is too large, the enemy's sword might hurt the character even though it passes by visually.

Other defects

When working on a major project, each stage of model production goes through many approvals and confirmations. It is tested by artists, modelers, and texturers, among others. Even then, testers can still detect visual defects that have appeared during the production process.

Inconsistency with the historical prototype

Players may accept unrealistic armor with giant shoulder pads and an implausible appearance of a flying machine in fantasy or science fiction games. However, if the game claims to be realistic, then there are certain to be meticulous players who will indicate that the propellers of certain aircraft models were located completely differently, and there were much more guns on certain battleships.

That is why it is a good idea for the tester to get acquainted with at least a couple of photographs of the tested object, if possible, in order to imagine what it looked like in reality.

Objects hanging in the air

A large number of specialists can work on a location at the same time. Some add or remove objects; others change the geometry of the terrain (map) or fix tiles (textures that are applied to the terrain). Because of this, defects appear when a previously existing object is drowned in a terrain or hangs in the air, because its old "support" was removed.

Visible joint between textures

Such defects are most often found on the terrain of large maps and locations when several textures are mixed. For example, a "seam" might be visible at transitions between a grass surface and sand or stone. For more details see Chapter 3.1.

Destruction of objects

Typically, the model has several states in order to add realism to different situation, such as original, damaged and destroyed. If the game assumes realistic destruction of objects, then a special crash model is created for them, possibly even with its own collision model. At a certain moment, the original model is replaced with a destroyed one.

Here it is important to ensure that the effect of destruction corresponds to the model itself. The destroyed truck should be different from the original one, and the pieces of a broken brick building must not be wooden.

Lighting defects

Lighting defects include:

- Defects in global illumination,
- Defects of point light sources,
- Defects of searchlight light sources,
- Defects of sources having an area,
- Defects in directional light sources,
- Defects in emitting light sources,
- Defects of diffused light sources.

Modern video game engines include very advanced capabilities for creating different types of light sources in full accordance with the laws of optics. However, there are still games where the lighting is not ideal, e.g., due to wrong lighting settings or lack of developer's experience. Consideration of visual composition (position of light, its angles, colors, field of view, and movement) has a large impact on how players perceive the game environment, and creates the right atmosphere emotionally to engage the player. Designers should take care of this by creating visual integrity. [Lee16], [Tavakkoli18], [Romero19]

Animation defects

This defect results from snapping the skeleton to the model, called skinning. The more bones the model has, the more realistic animation can be created. Different objects can be animated, e.g., the hair of the protagonist.

The bones of a model are dependent on each other. Therefore, when a hand is displaced, the bones of the palm will also move. All subsequent animation depends on how realistically everything was done and set up during rigging and skinning.

If the designer made errors, it could lead to lengthened arms/legs of characters in various animations, and parts of models that are "detached" from the rest of the elements. Often, these failures appear when colliding with other models or in various animations of the model.

Visual effects (VFX) failures

Games often use various effects related to the actions of characters, events and natural phenomena, such as explosions, sparks, smoke, disappearance and appearance.

When working with animations, visual effects are usually attached to bones, helpers (i.e., utility objects that are used to create and animate models), and objects in the scene. Creating visual effects is an iterative process where animation teams and visual effects artists interact. The latter may be asked to make changes or additions to the animation, such as,

- add more keyframes for a smoother trail from the sword,
- rotate the helper to use its rotation as a direction for blood splashes,
- change the camera switching to avoid the ugly look on the particle mesh.

First of all, it is important that such effects are synchronized with the events that produce them. For example, fire and smoke from the muzzle of the gun must be synchronized with the moment of firing and the bullet coming out of the barrel. Otherwise, the realism of the event will be violated. Such occurrences are traditionally considered a failure.

Another problem that causes VFX failures is non-compliance with the technical conditions associated with maintaining the frame rate and optimizing the hardware resources used to correctly display the effect. No matter how large-scale the effect, it is unlikely that the player will be happy to see twitching dust clouds or explosions stuck in time on the screen. As a result, special effects in games are burdened with strict technical limits, the purpose of which is to maintain a certain stable number of displayed frames.

When testing VFX, the tester needs to ensure that the hardware capabilities of the gaming device are being used optimally and that the effects are in sync with the events that trigger them.

It is desirable to find as many defects as possible at the model production stage. For example, exceeding the number of polygons of the model or texturing defects, which significantly increases the load on the hardware used in model calculations. This, inevitably leads to long delays in the game due to the overload of the video adapter processor.

In order to test this, the developer must have received predetermined and fixed requirements for aspects such as LoD and collision models.

Testers, as a rule, find manifestations of these defects, including the breakdowns and failures caused by it. If an object has, for example, a collision model with too many polygons, the defect will not be detected as long as the frame rate is high during interaction with this object.

3.2 Approaches to Testing Graphics in Game Products

3.2.1 Artistic testing

In the process of creating a graphic object, it goes through many different stages and reviews. Some people create the design and others put it into the application. As a result, the output may not be quite what the author of the design intended it to be. A tester can test the compliance of the design with the standards, and notice defects, but there are defects that the artist will notice more easily. For example, insufficient transparency of the background or the application does not behave as intended when scaling.

Conversely, the artist cannot foresee everything and will notice the defect only in the running application, when it has already become a failure. Therefore, it is good practice to give the application to the designer for review after the implementation to help eliminate critical defects. In this process, the tester prepares the necessary environment, creates test accounts, describes defects after tests and is then able to test them or to send them to the artist for review.

On projects where there is a large amount of content, there are often specialists who are involved in the graphics testing. These are primarily artists who thoroughly understand the process of creating models. They make a confirmation of each of the following stages of the model creation:

- Geometry creation,
- Creation of textures,
- Collision model creation.

The goal of this work is to improve the quality of the models before exporting to the video game engine. For example, they test aspects such as polygonage (i.e., the number of polygons) in geometry, the correctness of its location, the correctness of the unfolding of the model, the presence of elongated polygons, and seams in the model.

Artistic testing is a large, complex task that can be performed at various stages of production, starting with the creation of the simplest geometry and textures, and ending with exporting the model to the engine and placing it on the map. To conduct artistic testing, the tester often doesn't need special tools and content editors. It can be directly performed whilst running the game. However, the availability of tools and content editors allows defects to be found much faster and much more efficiently.

Artistic testing is performed by:

Role	Responsibilities for artistic testing
Artists:	When reviewing objects.

Role	Responsibilities for artistic testing
	When viewing and evaluating objects
3D modelers	When reviewing objects
Supervisors	When viewing and evaluating objects
Testing engineers	After the final export of models to the engine
Players	When participating in playtests

3.2.2 Technical Testing

Technical testing involves a set of tasks related to technical parameters of graphics, such as:

- Compliance with the limits on the number of model polygons, which were mentioned earlier
- Texture formats,
- LoD switching distances, for which it is enough to compare available resources with the specification.

Technical testing itself is performed by testers as well as by technical artists. It includes the procedures as described below:

Search for unused and temporary files in graphic content resources

When working with artists, resources may accidentally be added to the repository that will not be used later. For example, a developer might load to the repository all the resources, including unused ones. In this case, the amount of space occupied on the hard disk by the game client increases, which reduces the performance and eventually increases the total amount of patch delivered to the player (e.g., updating individual video game files). To prevent this, a search for unused resources is performed.

Search for the presence of all required content records in the client's resources

In the game client resources, graphic content does not exist independently of other resources. References to textures are written in models, references to models are written on maps. Even the smallest defects in such records can lead to unexpected consequences.

Testing texture formats

For each type of texture, technical artists set their own requirements. This is due to the need to strike a balance between the performance of content and the visual component. The texture format implies the type of compression, and its size.

Testing client logs when testing content

Sometimes the work of the client subsystems is not very noticeable. For example, post-effects are effects that are superimposed on the image over the image, such as a lens when looking at the sun. A dedicated texture is created for each post-effect, in accordance with the superimposition on the screen. When looking at the sun with a lens, for example, one will see the lens and rays of the sun, regardless of whether the texture is assigned for the post-effect or not. In this case, the player simply will not see the defect, despite

the fact that it exists. To find such defects, it is necessary to examine the client's logs because they will certainly mention that the render tried to use a texture that does not exist in the resources.

Verification of compliance with the limits on the number of textures, models

Each type of content has its own limits which usually are general for a specific group. For example, for indestructible multi-storey buildings, it is allowed to use up to 15k polygons, but for one-story buildings only 10k. Due to the fact that content edits are ongoing and the number of downloads to the repository is large, after each new export of the model, it is necessary to test compliance with the limits and requirements in order to prevent situations associated with a lack of hardware resources necessary for calculating the video processor models.

Despite the simplicity of the tests, it is vital to understand the importance of technical testing. Graphics can have a huge impact on overall system performance. Therefore, technical testing includes performance testing. to ensure that the performance and size of the game remain at the desired level. By increasing the number of textures and models the size of the build also increases. This mismatch of texture formats worsens the frame rate indicator, which leads to an increase in the memory consumed by the client. Technical testing makes sure that artistic content will be delivered to the player to the fullest extent and scope.

3.2.3 Gameplay Testing

Gameplay testing is a test approach aimed at those factors that affect the gameplay. This can be both an assessment of the compliance of the collision model with the visual model, and a verification that the settings of gameplay objects comply with the requirements of the game mode.

Gameplay testing of objects includes:

- Testing the compliance of Health Points (HP). This is value in role-playing and computer games that determines the maximum amount of damage that can be applied to an object to destroy it. Despite the fact that HPs are assigned to objects automatically, taking into account the material, size and type of the object, defects can still occur. Therefore, the purpose of HP is tested manually for the characteristics of the object.
- Testing that the collision model matches the visual model. As previously mentioned, the collision model is significantly simplified compared to the visual model. This may lead to defects when the player is visually hidden behind the model, but in fact can be damaged by an enemy.

Graphics testing is a complex process and its types overlap with each other. This test approach described above ensures that the graphical content of the game is thoroughly tested at various stages to detect graphics defects.

3.3 Graphics Test Execution

3.3.1 Graphics Test Execution at Different Stages of Object Production

Creation of a grey box

This is a simplified model of a graphic object, which is actually a mock-up used to test gameplay. At this stage defects discovered are primarily related to the convenience with which artists can work with this model.

Creation of visible geometry

The next stage of modeling and mapping includes mapping and creating a 3D projection onto a physical object of the environment. This takes into account its geometry and location in space. The visible geometry

and UV-unfolding of the model are created and the correspondence between the coordinates on the surface of the 3D object (X, Y, Z) and the coordinates on the texture (U, V) is checked. Since inspections are often carried out using 3D graphics editors, defects at this stage may also be detected by artists and 3D modelers. As a rule, these defects include:

- Duplicated geometry. Duplicating polygons can significantly degrade performance.
- Excessive detail. For example, a model that is out of the player's visibility has bevels, which means that when processing such “decorations” additional hardware resources will be allocated which the player will simply not notice.
- Insufficient detail. This type of defect is the exact opposite of the excessive detail defect. If the model is clearly visible to the player, it should be detailed enough so as not to look angular.
- Parts of geometry that are not rendered. For example, missing necessary parts from a model can render it unrealistic.

Texturing the model

At this stage, the textures are accepted by the artist or art director. In this case, the application’s correct application of textures, the absence of seams and pulling (i.e., incorrect polygon topology) may not be discovered. The test concerns only the general color palette and the level of contamination (e.g., presence of dirt, unnecessary color inserts, random pixels of different colors on the model). Typical defects found at this stage are cases where automatic mapping and subsequent texturing are performed in such a way that textures of different colors are applied to the parts of the model that have a common edge, which results in noticeable seams.

Review of LoD and collision model of the object

At this stage, both artists and level designers are involved in evaluating the quality of the models. Level designers evaluate collision models in terms of their impact on gameplay. For example, whether it is needed to use a detailed model at a specific location or whether it can be simplified. Typical defects in this case are models that or either too detailed (e.g., more than the allowed number of polygons in the collision model), or not detailed enough (e.g., contain an insufficiently detailed collision model).

Exporting model to video game engine

This considers the correctness of settings for the distances at which to switch LoDs and the number of health points of the object.

Testing is carried out by the testers directly in the video game engine, both in the content editors and directly in the game application. At the same time, most of the artistic tests are reduced to a visual assessment of the object, and a number of tests duplicate the tests of artists and modelers. Testing addresses the following:

- Visibility of switching LoDs of objects, taking into account the location and the situation in which the object will be used,
- Gaps and seams in the geometry of the object which are noticeable to the player,
- Visibility of inscriptions on various graphic objects,
- The destruction effect of the model (e.g., considering color and size.),
- Visual defects in object animation.

In artistic testing, players can directly take part who visually evaluate the content and can find other graphics defects.

Graphics testing is often done in conjunction with game maps testing. In this case, there are several more stages.

Placement of objects on the map

This is one of the most important stages for testing. Placement of objects is done by level designers. In this process, a typical defect that arises are objects "hanging" and recessed under the map (terrain). This often happens when many specialists work with the map in parallel who might change the geometry of the terrain, expose objects, and edit the map itself.

Creating and placing effects

At this stage, effects artists work on creating new special effects. Watching it in various situations allows them to detect artistic and technical rendering defects. An example of such a test is a visual test of an effect in an engine. Effects artists look at each effect in the game client to identify incorrect effect parameters (e.g., insufficient decay time of the effect).

Graphics testing is often done using playtests. This relates to any testing in which the map is tested in game situations. The goals of this testing are extensive and apply to any stage of production (e.g., doing an assessment of the gameplay on the map). Closer to the end of production, it can involve an artistic and gameplay assessment of the map from the players' point of view. In addition to collecting statistics that game designers need to adjust the balance on the map, such tests reveal various artistic and technical problems, ranging from hanging objects to graphics rendering defects (e.g., freezes, defects and failures). In purely artistic testing the playtest reveals problems that are noticeable from the player's point of view.

Testing the map

This is the final stage of testing where the map is completely ready and no more work is being done on it (with rare exceptions). Testers conduct a full map validation based on a map validation checklist, from looking for dangling objects to test animations. Testers focus on such tests, which allows them to detect many more defects (e.g., “hanging” objects) compared to artists. During artist tests, artists visually inspect maps, whereas testers use a specialized set of tools to detect more problematic issues in graphics objects.

3.3.2 Testing Graphics for Historical Accuracy

A separate area of testing graphic content is testing for historical accuracy. Historical accuracy considers both historical accuracy and historical authenticity.

Historical authenticity influences the quality of a game, in which a specified episode of history is described using characteristic images. These make no reference to specific events, personalities, etc.

Historical accuracy influences the quality of a game, in which a specified period of history is described. All key events are described in their dynamics, including all relevant personalities and their actions with images characteristic from the period. They fundamentally exclude images alien to the period.

As a rule, in games it is impossible to maintain historical authenticity, as the focus is on addictive gameplay. It is difficult to get players involved in a game in which the outcome is predetermined from the start.

However, historical accuracy is an area of gaming that needs special attention, especially when the game's plot is tied to a specific historical era.

Typical aspects to consider when testing graphic content for historical accuracy are:

- The resemblance of the characters to historical prototypes,
- Accuracy when reproducing architectural objects,
- Accuracy in reproducing weapons, equipment, vehicles, clothing of a particular era and their characteristics,
- Accuracy of recreation of everyday life,
- Accuracy of historical events and dates.

When conducting historical accuracy testing, the tester should have a wide range of knowledge and horizons. Very often, developers involve historical consultants in testing their products to create the most realistic game possible.

3.4 Tools Support for Graphics Testing

When testing content, graphics testers use a wide range of tools, ranging from in-game content editors to automated tools and test scripts. As a rule, games are created using video game engines that contain several built-in tools, including those for testing graphics: an object editor, a world editor and a lighting editor.

All of tools provide the ability to customize the existing content for the player to interact with it. Model editors allow binding effects to models such as various triggers, sounds and other events that happen when interacting with the engine. The world editor allows creating a map, placing objects on it and customizing the gameplay mechanics of the map. It is in these editors that testers do most of the tests. In doing so, they use the same tools as the developers. These editors also provide an extensive toolkit that allows the setting up the content and testing it.

Specialized tools are typically developed by graphics card manufacturers. These tools allow the capturing of frames in the game and detailed analysis of any application using various sets of APIs used to create 2D and 3D graphics. They also allow evaluation of the following:

- How the game frame is being prepared (e.g., geometry, textures, draw calls),
- Where performance problems arise,
- The geometry of the run,
- Debugging for 3D modelers, graphic artists and animators.

More details on how each of these tools are used is described in the specific software specification.

The graphics testing tools include automated scripts that provide automatic passage of the character along a given route in the game. They are especially used in performance and compatibility testing.

4. Sound Testing - 190 minutes (K3)

Video Game Specific Keywords

Ambient, binaural effect, distortion, lag noise (Foley), occlusion, reverb, skipping, sound discontinuities, sound effects, sound zones, volume

Learning objectives for Chapter 4

4.1 Features of the Sound Content of the Game Product
GaMe-4.1.1 (K1) Recall features of the sound content of a game product

4.2 Types of Defects in Sound Content
GaMe-4.2.1 (K1) Recall types of defects in sound content
GaMe-4.2.2 (K2) Classify the defects in sound content

4.3 Approaches to Testing Sound Content in Game Products
GaMe-4.3.1 (K2) Summarize the main approaches to content-auditory testing
GaMe-4.3.2 (K2) Summarize the main approaches for testing the mix of music and sounds
GaMe-4.3.3 (K2) Summarize the main approaches for testing music composition

4.4 Sound Test Execution
GaMe-4.4.1 (K2) Explain the levels of testing audio-music content
GaMe-4.4.2 (K1) Recall the features of integrating sounds into the client
GaMe-4.4.3 (K1) Recall areas of responsibility of sound testing
GaMe-4.4.4 (K3) Apply approaches to sound testing

4.5 Tools Support for Sound Testing
GaMe-4.5.1 (K2) Summarize the usage of sound testing tools

4.1 Features of the Sound Content of the Game Product

Sound is an important part of any modern video game. Sounds and music create the right mood, warn of danger, convey the emotions of the characters, complement and create a complete picture of the game world.

A person confronted with a movie or video game that has no sound at all is likely to experience slight dissonance. This happens because your brain is trying to activate your hearing, but it cannot, and instead signals an error of perception about what is going on in front of your eyes.

Low-quality, unrealistic, inappropriate or poorly functioning sounds can irritate the player and break the immersion in the gameplay.

The process of creating a soundtrack in a video game is quite lengthy and time-consuming, since it is necessary to take into account the content organization complexity of the video game:

- Environmental sounds,
- Character voices,
- Background soundtrack,
- Sound effects,
- Various sound objects,
- Voice files.

This complicates the task of testing

4.1.1 Types of Sounds

The sound design found in games can vary from video game to video game and be created for different purposes. Consider what types of sounds it is customary to distinguish, what functions they perform and for what purposes they are created.

Music

The main musical theme of the game, its calling card, can appear even in the simplest game. Tunes from old 8-bit video games have long been classics.

The main functions of music:

- Emphasizes the ‘epicness’ of the moment,
- Enhances the dynamics of the event,
- Increases the atmosphere,
- Puts the player in the right mood and explains the emotional state the character,
- Subconsciously forms a certain emotional response in the player due to the repetitive melody.

Sound effects

If the game world is filled with objects with which the hero can interact, each of them must have realistic sounds. It makes the game world come alive and believable.

Opening doors, blows, shooting and reloading weapons, the use of the first aid kit, the explosion of fuel canisters, the sound of breaking glass - all of this is necessarily voiced and played at the right moment.

Even in small games, developers add sound effects interaction with interface elements (buttons, switches, menu items). This gives the player feedback and an understanding that the player's actions are perceived and analyzed.

Character Sounds (Foley)

The sounds made by the characters in the game include the rustling of clothes, breathing, exclamations, steps, groans, etc. The term is named after Jack Foley, one of the pioneers of sound effects. In some games, such as shooters, such sounds can be an indicator that the character is wounded. This will be even clearer to the player than the pop-up damage numbers.

Speech

Although the speech is reproduced by the characters, it is still better to highlight it separately. In today's games, each non-player character (NPC) has a unique voice, recorded by a professional actor. The developers put a lot of money and effort into the voice acting, because quality recorded lines can correctly convey the emotions of the characters and create an emotional connection between them and the player.

Ambient sounds

Ambient sounds refer to the sounds characteristic of a particular location. Like the musical theme, ambient emphasizes the atmosphere and creates the necessary mood for the player. Ambient infers no objects from which this sound comes. It is an environmental sound that simply accompanies a certain location, situation, or stage of the game cycle.

Ambient sounds are usually played in the background and do not depend on the actions of the character. Examples of such sounds are:

- Rustle of leaves and birdsong in the forest,
- Splashing waves on the sea,

- Indistinct conversation of saloon visitors,
- Police sirens and the noise of passing cars, etc.

All this helps the player to navigate in space and better perceive the surrounding world of the game.

4.1.2 Sound Effects and Technology

Soundtracks are often processed in sound editors to create an authentic effect, taking into account the capabilities of modern audio equipment and technology.

In order to immerse the user in the virtual world of the game, using just realistic sounds may not be enough. To enhance the realism of what is happening, developers often hyperbolize some of the sounds. This technique is often seen, for example, in movies with scenes of fights, where the blows and even the movements of the fighters sound much louder than in real life.

To achieve this effect, various techniques and technologies are used.

Occlusion

One of the most common techniques is the occlusion effect when sound passes through a barrier (e.g., a building wall). For more realism, sound specialists in such situations don't just turn down the volume, but add a special effect.

The main sound and reflected sounds are realized differently when there is a blank wall between the sound source and its listener, when objects are in different rooms but in direct line of sight (doorway), and when objects are in the same room but cannot see each other (column between them).

Reverberation

Reverberation effect is used to convey volume and depth of space (from reverb).

In this case, the sound from the source in an enclosed space is reflected from the walls and causes numerous echoes, which gradually fade out. This effect works well if the character is in a cave, small room or other room with well-reflecting sound surfaces.

With advanced sound engines, one can easily create echoes and add volume to the sounds of gunfire, explosions, voices, footsteps, etc.

Binaural effect

The binaural effect (from Latin bini - two, pair and auris - ear) is based on the fact that a person listens with two ears simultaneously, and when a person turns head to the side, the sound reaches one ear before the other. Thanks to this, a person can determine in which direction the source of the sound is, and even the distance to it.

Special microphones are used to record sound with this effect, and for its perception it is recommended to use headphones. This is necessary so that sounds can come into each ear from a specific direction. The sound environment makes the immersion deeper and may help the player. Because of binaural audio, the player knows which side the sound is coming from, (e.g., where the enemy will appear in a moment).

4.1.3 Sounds Area

In some projects, so-called sound areas are used to distribute the sound environment in different locations.

In games, an example would be when a character in a role-playing game enters a village and the characteristic sounds appear, or in a strategy game the player brings the camera close to their base and hears the sounds of resources being gathered.

Sound zones must have transitions. Near the transition, the sound of one zone should fade slightly, and the sound of the other zone should appear.

An example of the effect of sound zoning in real life can be observed by opening and closing the windows of your room. When opening the window, the sounds of the street are clearly audible, and when the window is closed, they either are not heard at all, or they are extremely muffled.

4.2 Types of Defects in Sound Content

There are three categories of sound content defects:

- No sound,
- Wrong sound,
- Incorrect sound playback.

Missing Sound Effect

No sound is played when it should be, for a certain action of the character or at a certain moment.

Playing the wrong sound

Incorrectly set the sound of the object or the environment, for example, a cannon shot sounds like a pistol shot, a motorboat sounds like a plane, or characters (both in- and out-of-game) say someone else's lines

The sound effect violates historical authenticity, for example, game characters use modern vocabulary, movie and music quotes, the T-34 tank sounds like a T-72 tank. Rare, but can be meaningful if realism and authenticity are stated as a key value of the game under test.

Incorrect playback of the correct sound

Audio drop

When such a defect occurs, some of the reproduced sounds may disappear, as when you're on the phone with a bad connection. Because of this, the player may, for example, misunderstand the meaning of a phrase in a dialogue. If this is the only sound that is currently playing, then testers will easily find the problem. But if several sounds are played at the same time, and one of them disappears, then it is much more difficult to detect such a defect.

Skipping

Skipping in the sound of music or audio effects can be associated with more than just damage to the sound file itself. Skipping is often caused by performance problems. In this case, they occur when the frame rate fails.

Distortion

Due to performance problems, some phrases may sound distorted and inaudible.

Playback Lag

In games, there are moments in which the sound design lags behind the animation, object, or game situation displayed.

Examples of defects in the sound content and their possible causes

1. Lack of sound of the object/environment.

Defect: When moving on a metal surface there is no sound of footsteps, the character moves silently.

Cause: The developer forgot to set up or connect the sound on this surface.

2. The sound of the object or surroundings is too loud/silent.

Defect: A certain object in the location is burning, but the noise from the fire is too loud, or on the contrary, very quiet, which looks unrealistic.

Cause: The volume of the effect is set incorrectly.

3. The sound of the object or environment is set incorrectly.

Defect: While driving a motorboat on the water, the game plays the sound of a racing car.

Cause: An incorrect sound file for the object was written in the code.

4. Incorrect positioning of the sound depending on its source.

Defect: The game has a blacksmith who continually strikes their sword with a hammer, but the sound of the strike is played in another corner of the room.

Cause: In the sound editor, the sound effect is set to the wrong position.

5. Sound coverage area incorrectly configured.

Defect: The sound of a gunshot is heard only within a radius of a couple of meters. If the player moves further away, then the sound is not heard at all, which is not realistic.

Cause: The sound radius is set incorrectly.

6. Sound distortion.

Defect: Players experience the sound beginning to crackle and hiss.

Cause: A corrupted audio file is being used.

7. Continuous repetition of the sound.

Defect: The sound is looped and played continuously.

Cause: A pointer when the sound loop should stop is not set correctly.

8. Sound distortion in the form of "stuttering" playback:

Defect: Distortion of the sound design of the movement occurs during the movement of one of the game characters

Cause: A problem may occur due to problems with the sound assembly or due to the general instability of the game client.

9. Time delay of audio playback.

Defect: During a cutscene, a character starts to speak, but its line is reproduced a little later than when his lips begin to move.

Cause: the animation of the character's lips moving is out of sync with the start of the corresponding sound file.

4.3 Approaches to Testing Sound Content in Game Products

The sound design of the game may be represented in the form of effects, character voices, or music. Testing of games audio content may be performed in more formal or informal ways. Formal approaches to

sound testing include testing of presence, correctness, volume and integrity of all audio files. Informal sound testing is a sort of evaluation of the realism or maintaining the atmosphere of the game. For example, "heavy" music accompaniment to the location in games of the horror genre to accentuate the situation and periodically maintain the player in a state of tension.

The tester derives test cases from the previously reviewed specification of a particular sound object. From this specification the tester should unambiguously understand at what locations, game scenes and situations the sound should be played. In some cases, this may require a consultation with an audio designer to learn specific details about the sound content created. In other cases, the tester can use the video game resource editor to test the synchronization of sound files with the actions of the player or the environment in the game.

Testing the sound design of the game by a tester may involve several steps.

4.3.1 Content-auditory Testing

The tester listens to the sounds and tests the audio parameters of the created content using the sound editor or the video game resource editor. All sound files must be tested.

This is one of the most extensive stages. The following sound characteristics are checked against requirements:

- Appropriation to the object and the setting. For example, if the character is wearing iron armor, then the sounds of the metal should dominate the sounds of the fabric as he moves. The faintly audible explosion of a stick of dynamite or a fallen rock, the obvious stomping on the sidewalk when moving on the sand, can disrupt immersion in the virtual world.
- Voices to the actors. The voices of the actors in all-voiced phrases must correspond to the characters who speak them and to the game's setting. Usually, the voice must match the gender and age of the actor. It's done to maintain the general atmosphere, for example, the Southerner character has a Southern accent, and not modern Irish.
- Not annoying. Samples of multiple sounds together may sometimes become unpleasant to the human ear.
- Volume level. The volume for each sound file must be the same when using multiple files within one game.
- Sound effects. Effects applied to both the sounds and the voiceover must also be used correctly. If the location or scene is filled with various details, then the details are emphasized by voice acting. In a large office a person can hear the humming of the system units, the buzzing of the printers, the noise of the air conditioners, the pressing of the keyboards, and the conversations of the workers.

Careful dubbing of all objects contributes to the creation of a real picture. For example, when the player finds themselves in the village, they will not necessarily hear the mooing of cows, but they will most likely expect the characteristic sounds of a sawmill if one appears in front of them. A small stream in the woods may well have no sound design, but if the hero stands on the location next to a waterfall, the lack of water noise can only be explained by a defect (or turned off sound in the settings).

4.3.2 Testing the Mix of Music and Game Sounds

The size of the project and the professional skills of the development team vary from project to project, which makes it impossible to clearly separate the functions performed by roles. There may be situations where a voice actor is not required, the work of a foley artist (a specialist in creating noise) is entrusted to an audio designer, or even one specialist himself creates or buys ready-made samples, ties them into the client and tests the resulting sound.

The role of an audio designer is to create sounds and record audio files. These audio files should be tested for different uses. The sound system of an arcade is strikingly different from that of a cell phone - and the audio must be tested. Console games should play correctly on both small, built-in TV speakers and home theater systems.

Even a high-quality and seemingly appropriate sound can cause confusion and irritation to the player if the final mixing is not done correctly. For example, the sound of the action lasts much longer than the action itself, or its volume is out of place.

The tester tests the finished sound picture and the correctness of the tinkering of new sound changes into the client, using the resource editor built into the video game engine or based on their own experience or provided documentation.

Since during gameplay the player hears a large number of sounds coming from different objects and music playing in the background, it is important that all the audio components are in harmony with each other. The tester should evaluate the sound component of the project and identify possible deviations, unrealistic points concerning the volume, together with quality and naturalness of the sound based on common sense. It is also important to assess the accuracy of the location of the sound source in relation to the objects themselves. For example, in the game, when a character opens a door, the player hears the sound of the door opening not from the door, but wrongly from behind the character's back. It is also important for the tester to be able to correctly assess the timeliness of the sounds in relation to what is happening on the screen.

4.3.3 Testing Music Composition

Composition or mixing is the stage of creating a final recording from individually recorded tracks. This is the next step after audio recording consisting of selecting and editing (sometimes restoring) the original recorded tracks, combining them into a single project and processing them with effects. Editing is often a stand-alone stage of work.

Mixing in games that use electronic music is the next step after creating audio content. The sound recording stage of an electronic project is most often absent. The line between creating and mixing electronic music is blurred; the project comes to mixing already partially mixed, because many virtual synthesizers already have template processing of different sounds.

As a result, a multi-channel project is output into a monaural, stereo or multi-channel phonogram, which usually gets its final form in a process called mastering.

After mastering is complete, testing of the mastered music begins. The tester should test the music component for defects, sound discontinuities, and interference and report any failures found.

4.4 Sound Test Execution

4.4.1 Levels of Testing Audio-music Content during the Game Software Development Lifecycle

Often sound testing begins late in development. While a developer creates content for a game (object/character models, maps, items, etc.), the sound component is worked out at a later stage. At the development stage an object may have no sound, but an object itself already has a finished model with customized animation integrated into the client and ready for testing.

The sound samples must be tested in advance by the specialists who created the sound (i.e., audio designers, Foley artist, sound effects specialists). They may take part in the testing of individual sounds and then send them to development to further integration into the client.

4.4.2 Integrating Sounds into the Game

After integrating sound design into the game, the tester evaluates the following characteristics of the sounds:

- Ability to turn on and off sounds and/or music.
- Ability to change the volume.
- Compliance with a single format of audio file names and their correct distribution into folders in the project.
- Location of sound sources, sound propagation distance zones on the game map.
- System performance with integrated sounds.
- Compatibility of the sounds on different audio systems: headphones, speakers, monitor speakers.

4.4.3 Areas of Responsibility of the Persons involved

The process of creating music and sound design in the project involves several different types of specialists, performing certain functions:

Role	Responsibilities
Composer	Involved in the creation of sounds and music for the game.
Audio engineer	Involved in the operation of sound recording, sound amplification and broadcasting equipment.
Sound engineer	Specialist who possesses the maximum completeness of sound expertise and determines the final sound picture of each object, game scene and the entire video game as a whole.
Audio designer	Creates content from the original audio files with recorded sound (samples) and adjusts it in the sound editor.
Foley artist	Makes the sounds that exist in reality more vivid and rich, and is responsible for creating sound effects that do not exist in reality. By combining the sounds of ordinary objects, they record samples and create from them the effects of, for example, a laser sword humming, a zombie growling, or a portal opening slam.
Voice actor	Plays with their voice in order to breathe life into a character who will have audible replicas in the game.
Developer	Integrates the created and customized audio content into the game client code.
Level designer	Distributes the sound sources by level and then binds the sound files to the sound sources.
narrative designer	Specialist who is responsible for story and narrative in computer game development. He/she works on the sound and music played during moments related to the development of the storyline (cutscenes, intros, dialogues, significant moments in the game etc.).
Game designer	Involved in the development of game documentation (e.g., a video game design document). This document describes the rules and features of the game in simple language. Thus, even before the engine is developed, the game designer develops a holistic vision of the game. During the development process, they also participate partly in the role of "consultant", approving the compliance of proposals / changes

Role	Responsibilities
	to the main idea of the project. They also personally participate in the testing of the game.
Tester	tests the finished sound picture and the correctness of making new sound changes in the client.

4.4.4 Procedures and Approaches in Conducting Test Activities in Sound Object Testing

When testing sound objects, the tester must implement the procedures and actions necessary to obtain complete information about how properly the sound is set up in a particular version of the game. The list of actions directly depends on what objects need to be tested. For example, for the final test of the sound design of the added weapons in the client, the tester performs a number of actions: adding weapons in case they are not initially available in the client, loading new content on the test map and the whole list of actions necessary to maximally test all the sounds attached to the weapon (firing, reloading, reloading again, clicking to switch firing mode, etc.)

It is important to understand that the list of procedures and approaches for each object is different. For example, if an object of the environment is tested and not the object with which the character interacts, then the tester tests the inclusion of the sound design itself - for example, the murmuring of a stream or river when the character approaches. The tester should understand where to find information about all available sounds for a particular object. He should also take advantage of the built-in resource editor and sound editor to test the integrity of the sound content and synchronize it with the actions of the character and the game environment.

4.5 Tools Support for Sound Testing

Video game engine audio editor

The tester does not always have access to the video game engine's audio editor. Therefore, while testing a standard set of software for content validation is used. Such an editor is used by testers working within the development team (if any), or by specialists working directly in the audio editor (e.g., composers, audio designers, etc.). Audio designers have to participate in the testing. They are testing the assets including balancing levels, adjusting equalizers, and mixing/aligning all the assets both together and individually. As soon as the assets are added to the working version of the game, the audio designers should be able to get them easily and edit them if necessary.

To ease the maintenance of the sound contents, all sound samples should be on different soundtracks, arranged in folders, named, time-stamped and versioned. It is also effective to convert audio tracks to compressed digital format, so as not to burden the hardware resources of the sound adapter.

Map/location editor

Sometimes the tester gets access to the development software, most often the map/location editor. This allows the tester to expand their capabilities when testing audio content and all content that is in the client. Game developers add functionality into map editors that supports “simulated” gameplay directly on the level map. Thus, the tester can carry out the necessary tests without switching from the map editor to the game client itself, which significantly saves time.

Most of the sound effects and sound sources available on the map are arranged and adjusted using the level editor. In the map editor, the tester can see the location of sound sources on client locations, tied to different objects in the location, or characters. Accordingly, when specialist tests the sound source through

the editor, he can test the sound being played and the sound propagation distance, either displayed as a sphere, or as "zones" where the tester or end user begins to hear the sound/music being played, coming from a tethered sound source, when crossing their boundaries.

If the sound propagation is implemented in the form of "zones", the tester tests the sound playback from the source when crossing the boundary of the "zone". It is also relevant to test the volume rise while approaching the sound source to make the tied sound more realistic. The sound propagation distance test in the form of a sphere differs only in the shape of the propagation zone. The editor also allows testing of the binding, location and, respectively, the configuration of each client's sound source.

With this information, the tester can assess the realism of sound content, sound propagation distance, the correctness of the sounds (on the location, objects, characters) and indicate to the developer defects in case of detection. For example, create a description of a defect about an incorrectly set distance at a particular sound source. Or the volume of a particular sound (referring to a particular sound source) is over loud against the background of the "neighboring" sources. In the description of the defects, the tester, working through the editor, can facilitate the work of the developers, making it easier to find problem areas by pointing out a particular problem.

5. Game Level Testing - 65 minutes (K2)

Testing Keywords
Playtest

Video Game Specific Keywords
Color coding, video game level, inaccessible places, level editor, level prototype, narrative, structural geometry, trigger

Learning objectives for Chapter 5

5.1 Game Level Design Principles and Concepts
GaMe-5.1.1 (K1) Recall the components of the game level
GaMe-5.1.2 (K2) Classify the defects typical of game levels

5.2 Stages and Execution of Game Level Testing
GaMe-5.2.1 (K2) Summarize the tests carried out at various stages of the creation of game levels
GaMe-5.2.2 (K2) Compare the areas of responsibility of specialists who take part in testing the game level

5.3 Tools Support for Game Level Testing
GaMe-5.3.1 (K2) Summarize the usage of tools for testing game levels

5.1 Level Design Principles and Concepts

5.1.1 The Term “Level” and Its Specificity Depending on the Genre of the Game Project

The game level is a separate area of the virtual world of the game, in which the player must complete a certain task: find a treasure, defeat all opponents, or just get to the exit. A general task required to finish a level is inextricably linked with location and often consists of many small tasks and quests needed to finish a level or side-quests, which are not necessary to finish a level. In many video game engines level are split by loading screen.

Level design is the stage of game development associated with the creation of maps, locations, tasks, missions and other environments for the levels of the game. Level design includes the appearance of the game objects and game mechanics, obstacles in the player's path, the game's storyline, and other elements that collectively create the intended game experience.

To create levels, special software is usually used - a level editor.

Game levels consist of many components connected to each other into a single whole. Among the main parts are the following:

Structural geometry

Structural geometry, or basic level geometry, is one of the most important and formative elements of any game space. It is the structural geometry that sets the terrain relief and the surface on which the game characters can move.

Structural geometry also limits the level space available for characters to move. For example, there may be impassable mountains or other terrain features at the boundaries of a level.

If a level has clearly marked entry and exit points, then structural geometry can provide guidance on where the player should go to get to the end of the level.

Depending on location where the game takes place, various elements can act as objects that form the level. For example, in the city: these are buildings, streets, bridges and underpasses; outside it, natural landscape elements: fields, hills, rocks, ravines and caves.

Game environment

The game environment objects are used by the level designer to create a more believable image of the required game space.

For example, a village street is made up of houses, outbuildings and roads, so the designer fills it with objects of different sizes. Large objects are distinguished such as - trees, fences, carts, medium-sized objects - wells, barrels, benches as well as small objects, such as plants in the garden, stones, etc.

Lighting

The use of various types of light sources allows for the desired atmosphere at the game level. For example, it directs the player in the right direction, attracts their attention, or hides them from opponents.

Sound accompaniment

Background soundtrack or sound effects played for various actions or events. Read more in Section "4. Sound Testing".

Game functionality

The gaming functionality includes a system of settings for organizing the gameplay at a specific level. The following types of settings are distinguished:

Settings related to the passage of the level. These include:

- points of appearance of characters and their opponents;
- conditions of victory and defeat;
- points of automatic save of the game;
- a system of triggers that launch certain game events, or scripts. For example, every two minutes an airplane flies over the battlefield, or when a character enters a building, a story video is played, or when a guard crosses the border of a protected area, the alarm is raised.

Setting up the objects that the player can interact with, for example, doors, pickups, destructible objects, traps, etc.

Level physical shell settings that prevent the character from getting out of the level or getting stuck in its geometry.

As a rule, the settings of the game functionality are hidden from the players and are visible only in the level editor.

5.1.2 Understanding the Types of Defects in Level Design

There are a number of defects that are most common when creating levels. Tracking such defects can be difficult, so they can remain unnoticed at the level for some time.

Some dedicated map creation tools have built-in functionality to detect such defects on maps. Level designers often use these tools in the final stages of creating a map or level. But in most cases, the best way to find defects on a map is to test it by experienced players who are able to detect the problem and report it.

A significant proportion of the defects found at the levels are related to the appearance and location of game objects, their lighting, collision model, etc. A detailed description of such defects is given in Section “3. Graphics Testing”.

However, the following defects directly relate to the design of the levels.

Geometry

Geometry defects are situations where the player's character is partially stuck in some parts of the level / map. As a rule, such defects are found on the edges of the map, ledges, slopes. The player guides their character into an object or walks close to an object and gets stuck in it. Often, with the help of jumping, bending and other movements, the character manages to get out of such a situation, but in some cases this does not help. In these instances, the player has to restart the level/map, return to the last progress save point, or use a special button/keys combination to move to the nearest location without collision defects.

For example, a game character can get stuck in the geometry of the map, unable to get out, or into areas that should be denied access. It is especially important to identify such defects in multiplayer games, where level defects can put one player or one team in a better position.

Another common situation is death when being stuck, namely, the character visually falling under the surface of the level / map and falling down. Ultimately, the character either dies and the level has to start again, or falls endlessly in the void, and the game has to be restarted.

Inaccessible places

In video games, especially on maps in multiplayer games, there may be intentional locations in which the player gets a better position than other players. For example, a machine gunner on a hill will have a large firing sector. Although the player who is the first to take such a position gains some advantage over opponents, this is an intention of the developer, a design element of the level, and not a defect.

But there are situations when such a place is created by accident due to an error by the level designer. According to the developer’s concept, this area should be inaccessible to players. However, a certain game character with the help of its unique features (e.g., higher jumps than the rest) may end up in it. In this way, the player can gain an illegal gaming advantage over their rivals. For example, other players will not be able to detect it, damage it, etc.

Complicated gameplay

Problems associated with the lack of goal pointers or the non-obviousness of the required actions to complete the level. For example, the player has defeated all opponents in the accessible territory, but cannot understand what else needs to be done in order to go further.

Game balance level

A lot of defects arise in setting the level balance. For example, bot opponents that are too complex at the current stage of the game, and which completely exclude the possibility of passing this level. In other games, several teams try to get to the desired point of the point map before the opponents. Here, the defect in the level balance might be unequal starting conditions for the teams.

Inappropriate restrictions and conventions

These problems do not affect the gameplay and do not provide a game advantage, but they degrade the perception and immersion in the game.

For example, the player sees that the character's path is blocked by a waist-high fence, but it is impossible to get over it. As planned by the level designer, this place is an area inaccessible to the player, and that's

how it is supposed to be. But from the player's point of view, this situation looks like a defect that ruins the immersion in the game. Similarly, the player will consider as a defect a situation when the character is locked behind bars and cannot get out, although he sees that the distance between the rods is large enough.

Narrative

Level defects associated with a violation of the general style of storytelling, the plot of the game. The impact of such defects is minimal on the gameplay itself, however, they negatively affect the user's immersion and the general perception of the game. For example, if a character finds itself in a city that, according to the plot, was subjected to a nuclear bombardment, the appearance of intact buildings in it will cause dissonance.

Changing the color coding of objects

Objects encountered by the user during the game have different properties. Some of them are decorations, and with others the player can interact. Color-coding objects with different properties is an important part of level design. Therefore, it is desirable that once adopted color coding does not change throughout the game.

For example, a player discovered a red barrel at the level, which exploded from a shot. If a player encounters such a barrel again, they will expect the same behavior from it. Therefore, all barrels that explode from a shot must always be of the same color (usually red).

Boxes that the player can break, doors that the player can open, rocks and ledges that the player can grab onto and climb over, etc. should be different in shape and / or color from other objects.

Otherwise, the player will be forced to spend more time understanding what to do next.

5.2 Stages and Execution of Game Level Testing

5.2.1 Basic Stages of Game Level Design and Testing

The game level is an entity that combines game mechanics, visual objects, soundtrack, artificial intelligence and other components. Testing approaches for each of these components are discussed in detail in the relevant sections of this syllabus.

Game level testing involves testing all these components together and is similar to system testing, when all components are considered and tested as a whole.

Testing of game levels starts from the earliest stages of their creation. The objects of testing, the nature of the tests and the procedure for their implementation, as well as the areas of responsibility of various specialists depend on the current stage of development of the game level.

Consider the main stages of designing and testing levels using the example of a 3D game.

Game Level prototyping (Designer Block Out/Grey Box)

At this stage, based on a previously developed sketch, a 3D model of the future level of the game is created from special objects. Only elements that directly affect the gameplay are used: grey cubes, spheres, cylinders and planes, which schematically form the layout of the game level.

Such monochrome, undetailed geometric shapes allow the structure of the level to be quickly changed, such as to remove or add huge pieces of location, and, if necessary, erase everything cleanly and start over. Sometimes at this stage, not all game mechanics are ready for use, so the designer has to implement them gradually, adapting the level to the constantly changing development conditions.

The layout is created taking into account all proportions and scales. To do this, the level designer, together with the game designer, forms a set of metrics - the sizes and parameters that the game character possesses. So, the game designer provides information about the distance and height of the character's jump, about what positions the character can be in (e.g., standing at full height, squatting, lying down) and how this changes the size of the figure, the maximum angle of inclination of the surface along which the character can move without sliding. These metrics directly affect the dimensions of objects and elements of the level environment.

After the level designer has decided on the size of the level and set the correct scale for objects, other functionality is configured to support the gameplay at this level. This includes the points of appearance of characters, points of operation (e.g., triggers) of game events, etc.

Game level development at this stage, as a rule, takes place in several iterations, during which testing helps to identify and eliminate problematic elements.

Game level prototype testing

Due to specified requirements not being finalized at the stage of prototyping, exploratory testing and other experience-based test techniques are considered to be the most effective.

On the created layout, the first playtests are held to test the gameplay on this map. The test participants conducting the first playtests can be both testers from the game software development team and outsourced testers. Receiving feedback from test participants, the developer can draw a conclusion about successful and unsuccessful solutions and, if necessary, changes, moves or removes objects on the layout.

Also, by conducting playtests on the prototype, the game mechanics are tested and how they are combined with the arrangement and size of objects on the model.

For example that the shelters located on the level are large enough for the character to hide behind them from enemies, that the width of the crevice in the surface allows the character to jump over it, or that the character can climb all the necessary ledges and doesn't hit its head on too low ceilings.

Separately, the desired complexity of the game, created by the layout of the level, and the right feeling from the gameplay in general have to be tested. For example, if it is assumed that at a given interval of the game there should be a shootout with many opponents, then the level should have a sufficient number of shelters. If it is supposed to fight against one strong tenacious enemy (boss), then the level space is limited and cleared of unnecessary objects.

Geometry prototyping (Art Block Out/White Box)

Temporary 3D models created by the environment artist replace the objects from which the level designer made the prototype, and additional objects are added. The main task is to work out the visual component of the level without changing the gameplay itself.

However, such a replacement may cause defects at the level that complicate the gameplay. For example, a temporary tree model overlaps the level output. In this case, the level again goes into the hands of the level designer. Now it is required to test again whether the mechanics are working correctly, whether the player can go everywhere and whether they see everything they need.

Testing the geometry prototype

Testing at this stage is carried out to make sure that after the artistic drawing on the level there are no defects that impede the gameplay.

Examples of such defects can be a temporary 3D model of a stone blocking the level passage, or a tree crown blocking the player's view.

To identify such problems, playtests are again used, where the game mechanics are tested to validate that the size and location of object models correspond to the required metrics and the correctness of the gameplay.

In parallel with this, a test of the appearance of objects, illumination, environment, etc. can be carried out (more details are given in Chapter “3. Graphics Testing”).

Making the final version

At this stage, the level designer integrates the software of the entire development team into a single whole. From all the content created, he collects a single play space. At this stage, decorative details are added to the level, objects become similar to real ones in appearance, purpose, and location. For example, papers, stationery, computer monitors appear on the desk, which the player can smash and throw off the table.

Testing the final version

At this stage, performance and compatibility testing is carried out. Performance of the level is tested on different devices, the number of frames per second is measured, the quality of displaying visual models of objects at different distances, lighting and shadows is tested.

As part of playtests, testers test the presence of collision models for all game objects, the correspondence of physical and visual models of objects, etc.

For example, testing of collision models proceeds as follows: the tester brings the game character to the game object, for example, to a large stone, and tries to “walk over the stone” from different sides. With a properly configured collision model with stones, the character's body will visually only touch the stone, but not break through it or completely come in into it.

5.2.2 Areas of Responsibility of the Persons involved

Various specialists work on the creation of a game level or map, and often these tasks are performed in parallel, which leads to the appearance of various defects.

Consider the tasks that are the responsibility of these specialists from the point of view of obtaining a quality product:

Role	Responsibility
Game level designer	Responsible for the geometry and shape of the level, the location of objects and trigger points, the size of shelters, etc. All of this should support the core gameplay and promote user engagement in the game
Artist	From an artistic point of view, all objects on the map should have the correct textures, lighting and other visual effects. This applies both to the appearance (correct placement of dark and light areas) and to the realism of the image. For example, a hut in the woods made of concrete blocks would look strange and out of place.
Tester	Testers, as a rule, work with the final versions of levels and maps, when all objects have already been placed and configured. The main task of a tester is to test if a player can play on a given map without encountering technical and artistic defects. Interaction with all objects on the map is tested, namely • Objects have a physical model (collision model),

	• Lack of invisible obstacle, • Impossibility of getting out of the map.

5.3 Tools Support for Game Level Testing

Various tools are used to facilitate testing of levels or maps, including 3D editors, level editors, and video game engines.

However, testers use the same tools as the developers that created the level.

The video game engine usually has functionality that allows customization of the map and objects on it for various tests and searching for defects. The in-house developed engine may be changed by the request of the testing team in order to increase the effectiveness of tests.

For example, as a result of changes in the geometry of the level surface, some objects may be suspended in the air or sunk into the surface. Such defects can be discovered by visual inspection of all objects, or, if the map editor allows, a special algorithm may simplify the test.

In addition, by disabling the display of textures of various objects in the editor, the tester can find impassable places on the map, or vice versa, a viable path unintended by the map designer. This is especially important for multiplayer games, because in some cases, this can give some players an unwanted advantage over others.

6. Game Controllers Testing - 95 minutes (K2)

Testing Keywords
Compliance, ergonomics testing, functional testing

Video Game Specific Keywords
Accelerometer, video game controller, gamepad, gyroscope, racing wheel, touchscreen, trackball

Learning objectives for Chapter 6

6.1 Principles and Concepts of Game Controllers
GaMe-6.1.1 (K2) Classify typical input devices and specialized ones
GaMe-6.1.2 (K2) Give examples of different input devices in terms of their application
GaMe-6.1.3 (K1) Recall different types of game controllers
GaMe-6.1.4 (K2) Classify the defects in a game product related to the specifics of game controllers, and possible causes of their occurrence

6.2 Approaches to Testing Controllers in Game Products
GaMe-6.2.1 (K2) Give examples for test conditions to be covered when testing game controllers
GaMe-6.2.2 (K2) Classify tasks for UX specialists, testers, and game designers during game testing

6.3 Tools Support for Game Controllers Testing
GaMe-6.3.1(K2) Summarize the usage of tools for testing the behavior of game controllers

6.1 Principles and Concepts of Game Controllers

6.1.1 Types of Game Controllers

A game controller is an input device that is used in console and computer games. The controller is usually connected to a game console or personal computer. Using the game controller, the player controls the movement and actions of the game elements. In this case, the type of elements depends on the game itself, but most often it is one of the characters in the game.

Typical input devices

Gaming devices, such as phones, PCs, consoles and slot machines are guaranteed to allow the use of one of the following devices.

Input Device	Description
Gamepad	This is the primary input device in game consoles
Keyboard and Mouse	Since these devices have become common input devices for personal computers, they are also commonly used for computer games. Some game consoles also allow a mouse and keyboard to be connected and controlled in games.
Racing wheel	is a controller that has a knob for rotation and one or more action buttons
Trackball	looks like a ball half protruding from the base; the ball is rotated by swiping the palm over it
Touchscreen	used in phones, PDAs, portable consoles and modern slot machines. Some games have gameplay elements specifically targeted at the touchscreen

Specialized devices are devices focused on certain types of games.

Specialized device	Description
Joystick	This was originally a universal gaming device. A keyboard / mouse or game pad was found to be preferable in fast-paced games, and the joystick became a specialized device for games in the flight simulator genre. However, out of habit, gamepads are often referred to as "joysticks".
Racing gamepad	Used to simplify the game of racing on the console. In fact, this is a regular gamepad, which has an additional steering axle built into it (and sometimes analog throttle and brake buttons). Such a device is much cheaper than a full-fledged racing wheel.
Steering wheel	For civilian flight simulators (military flight simulators use a joystick).
Pedals	For car and flight simulators that have fundamentally different designs.
Computer throttle	Also known as engine control lever. Used for flight simulators.
Shift lever	For driving simulators.
Light pistol	For shooting objects on the screen.
Graphics tablet	Used to control the cursor instead of the mouse.
Rhythm controllers	Used in music games to simulate musical instruments such as guitars, drums, or a DJ's console.
Dance platform	A dance floor (like on slot machines). It is a platform with several buttons that can be stepped on with feet. The gameplay of such games is to step on the necessary sequence of buttons to resemble a dance.
Gaming keyboard	This is a specialized keyboard with buttons which are located in accordance with the specifics of a particular game, and additional buttons for creating macros.
Fishing rod controller	For fishing simulator games.
Microphone	A microphone or headset is used as an additional input device, with which commands are given to the video game such that characters and the players can communicate.
Train control panel	a simulator of a control panel for traction railway / tram rolling stock control.

Motion capture technologies

- Since the early 2000s, head tracking systems have been used to play flight simulators and for people with reduced mobility,
- Remote control devices that track its position in space using IR sensors and accelerometers,
- Cameras that track the controller's movements in 3D space and recognize images,
- Devices which allow the use of verbal commands, body postures, and displayed objects or pictures.

6.1.2 Defects Related to the Specifics of Game Controllers

The causes of defects associated with controllers can be different. The appearance of defects can be caused by the software itself, the marriage of controller components, and even the developer's failure to comply with the instructions for using the controller from the manufacturer:

- Outdated controller driver,
- Controller model incompatibility with application,
- Defect of an individual device or the entire batch,
- Inconsistency of the gameplay with the instructions.

The most common defect is the lack of replacement or the complete absence of a tooltip when switching controllers during the game. Key bindings may vary from game to game. They can also be reassigned by the player at their own discretion.

An outdated version of drivers or their absence can lead to the fact that the controller does not work as expected. If software defects can be eliminated by updating, then technical malfunctions and shortcomings of controllers can be corrected only by releasing their new revision.

However, an inaccuracy in reading the movement of a racing wheel or any other controller can arise from a hardware defect or a defect in software calculations. For games where the accuracy of reading the control signals of the game controller is critical, the video game design document must contain the required values in degrees of inclination.

Also, when releasing a video game on a popular platform, the platform owner can provide requirements for the in-game images of their controller. In-game images are the controller images used in the game. For example, it can be a schematic representation of a gamepad in the key binding settings menu or in tips for gameplay. Images can not only be inside the game; controllers can also be displayed on software packaging or on a digital cover. This requirement usually applies to well-known publishers and simultaneously manufacturers of consoles and controllers. Corporations, in their test documentation for game developers, may provide requirements for how controllers should be depicted in an application, including both outlines and trademarks.

Also, for video games where the accelerometer and gyroscope of the controller are used, the platform owner imposes security requirements. The simplest example would be the need to make waving with controllers to implement the gameplay. The motion sensors inside it allow it to be used as a control in 3D space. In such cases, it is required to indicate before starting the game, that the user needs to put on the holding straps that are attached to the controllers. Otherwise, by making sudden movements with the controllers, they can slip out of the user's hands and damage the surrounding equipment or, even worse, become a health hazard.

6.2 Approaches to Testing Controllers in Game Products

Testing using game controllers usually begins at the moment when the game functionality is ready for using some standard controller: for PC it is a keyboard / mouse, for consoles it is a gamepad.

This type of testing may include the following:

- connecting / disconnecting the controller from the PC / console,
- low battery level of the controller,
- game support for certain controller manufacturers,
- support for certain APIs that allow an application to receive data from a controller,
- the use of one and / or several controllers,
- non-trivial use of the controller (negative testing),
- vibration (presence and degree of its intensity).

Functional testing

If the software requires the use of a controller as an input device, then all the functionality of the application must interact correctly with the connected device. The user should be able to control both the elements of the user interface and directly the gameplay without changing the controllers. Assigned control elements:

buttons, sticks, racing wheel turns, voice commands or movements in space must correspond to the actions of the game character. A tester also needs to make sure that when the controller is disconnected from the device, the application pauses, if possible. If the implementation of the gameplay is carried out in real time and cannot be stopped, then disconnecting the gamepad should not lead to the disconnection of the player from the server, and also be accompanied by an information message.

Security testing

A controller may contain technological vulnerability and open access to hacking the console. Security testing must be performed to mitigate security risks: gaining unauthorized access to developer mode, avoiding blocking the device over the network by using flight mode, etc. [ISTQB_AL_SEC]. Security testing of game controllers mixes approaches of discovering hardware faults which influence software functionality.

Ergonomics testing

The buttons on the controllers of the most popular brands have a well-established purpose. The X, A and □, X buttons on gamepads are very often used in applications as buttons for “accept”, “accept selection” or “interact” with an interactive item, and buttons B and ○ as “cancel”, “refusal” or “return”.

Button combinations are set up and controlled by the software. When testing button combinations, the tester needs to pay attention as to whether they are anatomically convenient for different groups of players (depending on genre, age, disability, etc.): whether the player has the ability to reach several buttons simultaneously or sequentially.

Compliance Testing of Game Controllers

UI / UX specialist and game designer roles are obligatory when developing games. Their task is to develop an interface that will be pleasing to the eye, be functionally convenient, and comply with generally accepted conventions, or standards of the genre. The purpose of testing will be both to test the operability of the interaction of the inputs sent from the controller and the actions performed in the interface, as well as to test the overall UX (user experience).

The tester needs to make sure that the developer is using traditional layouts for popular controllers, for example, as previously mentioned, the X, A and □, X buttons are used as “agree” buttons, and the B and ○ buttons are used as “cancel” buttons. For the keyboard, the W, A, S, and D buttons are used as directional buttons, Ctrl or C for crouch or creep, and the space bar for jump. The traditional function of the left mouse button would be a shot or attack for first person shooter games, or a choice for strategies. The right mouse button is usually reserved for aiming in first-person shooters or moving troops in strategies.

Testers also need to make sure that the use of any controller does not give the player a significant advantage over others. Since gamepads are significantly inferior to the keyboard/mouse pair in terms of speed and accuracy of aiming at a target, they are often added auto-targeting to a hostile target and subsequent pursuit. The sight simply “sticks” to the opponent. In this case, it is the tester's responsibility to keep track of how far the crosshair begins to follow the opponent and whether it grabs their head, which is often a vulnerable area, inflicting additional damage when hit.

6.3 Tools Support for Game Controllers Testing

Several software tools may be used to support game controllers testing by video recording or schematic representation of controllers.

Capture/playback tools allow recording video from a PC screen. They are used for defect reporting to display the nature of the defect and steps to reproduce.

Another type of tool is used to display a schematic gamepad representation and commands on video. These tools simplify understanding of controller activities to discover and analyze the defect.

There are services which support testing of the controller input and deflection degree of gamepad sticks as well as controller vibration performance (see [URL3]). Such services have all the necessary testing tools for the gamepad, helping to assess controller health.

Special tools to display keystrokes on the keyboard are also available and may be useful during test execution.

7. Localization Testing - 155 minutes (K3)

Testing Keywords
Compliance, internationalization, localization

Video Game Specific Keywords
Cultural adaptation, historical accuracy, locale

Learning objectives for Chapter 7

7.1 Principles and Concepts of Localization Testing
GaMe-7.1.1 (K1) Recognize the localization test steps
GaMe-7.1.2 (K1) Recall the main objectives of internationalization and localization
GaMe-7.1.3 (K2) Compare internationalization and localization capabilities

7.2 Types of Localization Defects and their Causes
GaMe-7.2.1 (K2) Classify the localization defects and their causes

7.3 Localization Testing Approaches and Execution
GaMe-7.3.1 (K1) Recognize the full and partial localization testing
GaMe-7.3.2 (K3) Classify localization testing types
GaMe-7.3.3 (K2) Summarize testing tasks for a writer, editor, translator, and localization tester

7.4 Tools Support for Localization Testing
GaMe-7.4.1 (K2) Summarize the usage of tools for testing of game localization

7.1 Principles and Concepts of Localization Testing

7.1.1 Localization and Internationalization

The process of localization of any product, whether it is a video player, a game product or an operating system, has similar stages, approaches, techniques and differs in the unique internal procedures adopted in the software development company [Chandler11].

Game development is an idea that is implemented through writing code, supplemented with the necessary content and released to players. An important early stage in the development is to determine the market for which the game will be released:

- This stage is not always taken into account, which may negatively affect the sales of the game in the future.
- The presence of a local version of the game is mandatory for its successful promotion on the national market.
- Most games are developed taking into account the need to adapt the product to the target market.
- Adapting the game to the target market can increase its sales multiple times.

Since the release of the game to other markets may occur after the end of the development of the local version of the game (e.g., the company has money to launch the game in another market or partners have appeared who are ready to promote the product in certain countries), then when developing the game, the team must take into account that the game can be further adapted to some other markets. Otherwise, subsequent adaptations may require large financial and resource costs.

Internationalization

To avoid risks and difficulties of adaption for a specific region, a process called internationalization is used. Internationalization is the adaptation of a product for potential use almost anywhere, while localization is changes for use in a specific region.

Unlike localization, software internationalization is a set of activities performed during the development stage to facilitate subsequent translation and localization of software.

Internationalization is carried out in the initial stages of development and in most cases is carried out without the involvement of linguists-translators and is the responsibility of the software developer. Localization involves specialist translators (in some cases native speakers) with a large amount of additional knowledge.

Internationalization includes:

- Creation and development of software in such a way that there are no barriers to localization and international use. Possibly using Unicode or providing a character encoding approach (if required).
- Create opportunities to use elements that cannot be used prior to the localization process. For example, adding a framework for bidirectional text to the DTD (Document Type Definition) markup language. Or adding to the CSS (Cascade Style Sheet) a foundation for vertical text or non-Latin typographic characters.
- Ability to support regional, linguistic or cultural references. This usually includes the introduction of predefined localized data or a translation created earlier and stored in specialized software. Examples include: date and time formats, local calendars, number formats and number systems (Roman and Arabic numerals), selecting and presenting lists, and using personal names and contact forms.
- Extract localized elements from code or content so that localized versions can be downloaded later or selected based on user preference. As a rule, this is implemented in the form of loadable language packs.

This listing does not necessarily include the localization of the content, program, or product into another language. These are the techniques and approaches to software development that enable easy migration to localization in the future.

Localization

Localization is the process of adapting software to the culture of a country. A special case of localization is the translation of the user interface, documentation and related software files [Retsker81].

Game localization includes:

- Translation of texts of game dialogues and subtitles, tooltips, descriptions and messages, names of characters, names of objects,
- Translation of screensavers, interface and menu items,
- Redrawing textures and graphics,
- Selection of actors, dubbing and recording of sound files,
- Integration of localized materials into the game,
- Translation and adaptation of the project website and printing,
- Translation of advertising materials (news, press releases and marketing materials),
- Support for the game after the release of the localized version (updates, news and patches).

Thus, testing localization is a test of how well a game product is adapted for a specific target audience in accordance with its cultural, linguistic, religious, political and other characteristics and references. Typically,

cultural and linguistic aspects are considered here, in particular the translation of the user interface, documentation and files into another language, as well as the formats of currencies, numbers, times, phone numbers, etc.

Localization testing includes testing the content of a game application against linguistic and cultural requirements, as well as the specifics of a particular country or region. This type of testing helps to find localization defects or translation failures in a localized version before the final product reaches the user. The goal of localization testing is to find and fix defects in different localized versions of a product for different markets and regional settings (locales).

It is important to note that localization is not just translation into multiple languages, and localization testing and linguistic testing are not the same thing. Linguistic testing mainly consists of testing for spelling, grammatical and stylistic defects.

7.1.2 Difference between Localization of a Game Product and Application Software

The difference between testing the localization of game products and application software is based on the understanding that these two types of products differ significantly from each other in various aspects taken into account during localization.

The main differences are:

- Adaptation of graphic content to the target audience
- Localization of audio content
- Localization and adaptation of text content
- Compliance with genre and literary features

As a rule, a large amount of graphic content may require adaptation for the target audience. This type of graphics includes signs, characters, game objects, level maps, symbols and paraphernalia, game items, advertising screensavers, etc.

Audio content may also require careful localization. Often, the audio content of a game product must be duplicated by the actors in the language of the target audience, and the content itself must be adapted.

Mixed styles of text content require proper translation and adaptation. the following styles can be used in a game:

- A scientific style (description of mechanisms, instructions),
- Journalistic style (newspapers, magazines, articles),
- Artistic style (personal diaries, books),
- Official business (dossier, various documents),
- Colloquial everyday (dialogues of characters).

For example, in the genre of adventure (quest), there will be scientific, journalistic, colloquial every day, since the player needs to repair or assemble some device according to the instructions, find the necessary information in newspapers and interrogate people.

There is a requirement for the mandatory adaptation of the content of a game product, taking into account such factors as: historical accuracy, religious, cultural, political and ideological characteristics of the target audience, etc. In some cases, violation of this requirement can lead to an ambiguous perception of the product by the target audience and prohibit its use in the territory of a particular country or region.

There is a need to comply with genre and literary peculiarities when localizing a game product. For example, in multiplayer RPG, when translating, it is necessary to take into account the names of the players' races, names of objects, etc., accepted in the genre. Sometimes content adaptation is required using stylistically close and genre-related concepts.

Additional knowledge and information are required for the implementation of high-quality translation of a game product. For example, various references to other game products, media works or reality events that may be contained in the game.

Below are some of the approaches to the localization of gaming products underlie localization testing.

7.1.3 Localization testing stages

Localization testing includes testing the correctness of translated content, various interface elements, defects and system messages,and testing the Frequently Asked Questions and Help sections.

Translation testing

The purpose of translation testing is to test the multilingual interface of the game for translation defects, correctness of postal addresses, first and last names, currencies, date and time formats, etc.

Sometimes during development, it is necessary to adapt the appearance of the number and the numerical value to national standards (e.g., units of measurement). When testing, the tester should always remember which system is adopted in a particular country in order to clearly inform the user about speed, length, weight, temperature, etc.

The message “You are moving at 62 miles per hour” may be difficult for a player from a country using metric units to understand. In this case, it is not enough just to change the numerical value of the speed, but it is also necessary to adapt the unit of measurement.

The above also applies to the currency used in games for making purchases. The simplest example of price localization is the automatic translation into the currency of the region where the product was activated. In addition to conversion at the exchange rate, the corresponding currency symbol must also be present. However, when developing an application, the tester must also take into account the fact that software distribution services may set regional prices.

In various video games, in-app store prices are converted to currency and vary by region.

When testing games, attention should be paid to the graphics. They must correspond to the realities of the country for which the game is being published. For example, road signs may look different from country to country. In addition, pictures and scenes of local holidays are added. In Muslim countries, graphics are radically revised - all images of people and animals are removed and arabesques (complex oriental medieval ornaments consisting of geometric and plant elements) are added.

Celebration of the Chinese New Year is especially popular. In many games, the interface is decorated with national symbols: lanterns, dragons and fireworks. Also, the game may be replenished with limited (available for a limited time) content: stylized Chinese heroes of myths character skins, animations for launching firecrackers and rockets, riding pigs and dragons.

In games, jokes often have to be adapted, and occasionally even the plot has to be adjusted to align with the mentality of the country in which the video game will be sold.

Localization is not limited to working with text - it also includes the reconciliation of cultural characteristics and moral and ethical aspects.

Traditionally, there are some topics and directions in testing localization that testers should pay the closest attention to:

- Religion, including occultism and satanism;
- Sex, revealingly dressed characters and vulgar language;

- Prejudices and stereotypes associated with culture and the people themselves;
- Wars, military conflicts, terrorism;
- Politics, including the specific view of different countries on history.

For example, if a video game contains the presence of a passage from the Koran when voicing a video game, most Islamic countries will most likely ban this video game.

In the conditions of strict control of political correctness in the modern world, its observance should be taken seriously.

In addition to the difficulties caused by the technical component of the game or insufficient preparation for the localization of the game, a video game translator may also face a number of difficulties associated not with the game he is translating, but with the need to have additional knowledge to carry out a high-quality translation.

Additional knowledge also means other game products background, media works or reality events that may be contained in the game. Such references must be translated properly in accordance with how they were previously translated. Therefore, the translator must be aware of the latest events taking place in the world, as well as various popular films and games.

Compliance testing

The system of age restrictions for games and applications takes into account the peculiarities of the legislation and culture of individual countries. This allows developers to more accurately define content restrictions and distribute applications to the audience for which they are intended.

When testing localization, the legal requirements that are supported in different regions should be taken into account. For example, in Russia the age of majority comes at 18, in the United States, depending on the state, this age varies from 18 to 21, and in Japan, young people are considered adults from the age of 20. Games with different content in most cases must follow the age restriction system in force in different countries.

The same game may receive different age ratings in different countries.

Game rating in different countries
https://www.kaspersky.com/blog/gaming-age-ratings/11647/

When testing localization for compliance with legislation and legal requirements, it is necessary to take into account many circumstances, sometimes associated not so much with the target language, but with the legal and cultural characteristics of the country for the market in which the game product is offered.

In addition, it is also worth paying attention to the support in the game of the currency and operations with it for each country. For example, if in a certain country foreign exchange transactions are prohibited to everyone except the Bank of that country, trying to launch a game in that market will certainly lead game publishers to serious problems.

Indirectly, currency transactions include the purchase of loot boxes - literally “boxes with prizes”, which, as a rule, contain various character skins, consumable items, upgrades or mounts.

In December 2016 the Ministry of Culture of the People's Republic of China announced the enactment of a law requiring an online game publisher to publish the acquisition probability for all virtual items and services starting from May 2017.

In Australia, loot box games are subject to gambling restrictions if they can be played for money or anything of value. Questions remain even in the case when the value of an object that exists only in the game can be determined only in connection with the prestige of this object.

Colors and symbols

Specific colors and symbols, which may have different meanings depending on the country, should also be considered when testing localization.

For example, red for the inhabitants of China is a symbol of endurance and faith, in India it symbolizes purity. Europe, on the other hand, sees sin and sacrifice in this color. For the people of South Africa, it is the color of grief. In the United States and Japan, red symbolizes danger and terrorist threat, while the Egyptians associate it with mourning. Therefore, when testing a game project, the tester needs to pay attention to color schemes, since they can be important.

In addition, when testing, the tester should pay attention to support for regional keyboard layouts and hotkeys.

In the event that the application uses integrations with third-party resources (e.g., cloud storage or social network), it is necessary to take into account their availability for regions.

Thus, the testing processes for localization and internationalization will be different:

Internationalization	**Localization**
UTF encodings	Translation
Data formats	Legal requirements
Text direction	Currency and foreign exchange transactions
	Colors and symbols
	Keyboard layout and hotkeys
	Integration with third-party resources

7.2 Types of Localization Defects and their Causes

7.2.1 Possible Causes of Defects in the Localization of the Game Product

Even large companies are not immune from "cultural" flaws. When releasing any application, the tester should familiarize themselves with the precedents and existing regulations. In some cases, one phrase or joke can lead to a complete ban on the sale of the game in certain countries. The following causes of possible localization defects (e.g., details or contents prohibited in a certain region) must be addressed:

- Images,
- Sound and soundtracks,
- Realistic or historical scenes,
- Phrases or quotes.

7.2.2 Localization Defects and Risks

Poor or lack of knowledge in the target language or the product itself can significantly complicate localization testing, especially if the product is a representative of the simulation genre: medical, technical or sports [URL4].

It should also be borne in mind that testing localization can be a rather long process, since it takes time to study the characteristics of different regions.

Major localization defects can be grouped as follows:

Technical Aspects

- Mojibake (garbled text) / Garbage characters. Distorted text appears as a result of decoding text using an incorrectly selected character encoding. The result is the systematic replacement of characters with completely unrelated characters, often from a different writing system. As a rule, this leads to unreadable texts.
- The localized string does not fit into the boundaries set by the interface (cut off, scrolled). A term translated from one language into another may use a different number of characters.
- Offset. After an application is localized, the layouts of user interface elements may require reconfiguration to maintain the original alignment.
- No line translation into the localized language. Testers should pay attention to texts in the dialog boxes, images or screenshots in documents or the user interface. All of this content needs to be localized to meet user expectations.
- Overlap. This happens when certain controls in a window or dialog overlap with others.
- Missing texts. During translation or creation of applications, texts occasionally may be lost.
- Wrong font / size. Different countries use different default fonts and sizes. For example, Asian countries usually use a font size 9 by default, while American uses a font size 8. This issue usually does not interfere with any functionality, but it greatly affects the user experience, making the text difficult to read.
- Incorrect hotkeys. In some cases, the hotkey is not available due to absence of the letter in the localized language or keyboard.
- Variables in text templates. Different types of languages (analytical, synthetic) use different forms of declensions, cases and plurals, they may change or not change endings and sounds, thus variables can affect the forms of words. One of the most common errors developers make is dividing sentences containing variables into parts. A grammatical structure of the target language is almost the main cornerstone of localization. Indeed, when adding variables, developers often do not take into account that in a number of languages their meaning can change the construction of a phrase.
- Out of sync of the sound sequence of the original and the translation. When translating, the characters' phrases may be longer or, conversely, shorter than in the original due to different

number of syllables. Video clips in games are often pre-recorded, which forces localizers to take a more careful approach to the translation of cues: it is important to maintain a balance between the accuracy of translation and the number of syllables, while also taking into account the synchronization of sound with the movements of the characters' lips (lip sync).

Translation defects

- Incorrect translation or transliteration of proper names, dates, numerical values, names of historical events, holidays, realities, jargon, profanity, colloquial vocabulary, abbreviations, "evocative" names and nicknames, etc.
- Contradictory terms. The same term should have a consistent translation throughout the application. It often happens that translations are incompatible with each other.

Cultural adaptation of content

- Humor. The attitude towards objects of jokes, humor and satire, consideration of humor and its permissibility in relation to individual objects, etc.
- Religion. The attitude of the target audience to religious issues. including religious sites, rituals and ceremonies. Attitude of the target audience to marginal cults (Satanism, paganism).
- Historical accuracy and perception of events. The interpretation of historical events, military conflicts, discoveries in the field of science, the appearance and characteristics of specific existing objects, the peculiarities of peoples' way of life, etc.
- Peculiarities of national culture and worldview (including attitudes towards children, sex, violence, other cultures, etc.). National stereotypes, national cuisine, clothes, peoples' lifestyle. attitude towards children, attitudes towards sexual minorities, violence against animals, etc.
- Legal restrictions. Taking into account age ratings, legislation in the field of child protection, religious beliefs, propaganda of violence, drugs, sexual relations, race, terrorism, speeches against the government and the state system, bans on the use of specific symbols, etc.
- Excessive localization. Not everything needs to be localized and some elements must retain their original appearance without translation, for example: trademarks, logo, abbreviations, products names.

7.3 Localization Testing Approaches and Execution

7.3.1 Difference between Full and Partial Localization Testing

Full localization testing

This implies a thorough testing of the localization for defects. This type is effective only when developing a new localization for a client, provided that all strings have been translated and have not been tested earlier. It is the most expensive, as it involves all types of tests.

Partial localization testing

This is used when text changes within localizations that were previously tested. Validation identifies lines that were changed as part of a major update that affected the main story and its reference localization. Only the modified text is tested, which must match the text previously presented in the game. This test approach is considered optimal in terms of achieving efficiency while minimizing costs.

7.3.2 Procedures and Approaches for Testing Localization during the Software Development Lifecycle of a Game Product

Localization testing verifies the translation, supporting files, correct justification and adaptation of interface elements, as well as the rules for writing text.

The goal of localization testing is to ensure that the game supports multilingual interface and functionality, and that there are no localization issues (translation into another language, date and number format, mailing addresses, order of first and last name, currency, etc.).

Localization testing process includes:

- Determination and study of the list of supported languages,
- Testing the correctness of the translation, including user interface elements, system messages and defects,
- Verification of the translation of the "Help" section and accompanying documentation (if any).

Localization testing involves comparing the strings translated by the localization team with the strings of the reference localization to find:

- Grammar, punctuation, syntax defects,
- Violations of the regional requirements of the tested localization (format of time, date, measures, legal compliance, etc.),
- Lack of necessary technical data (variables, sections used to format the text, etc.),
- Violations of the art style and context of reference localization,
- Defects in the display of text in the game interfaces (too long lines, violation of the display format of fonts, etc.).

All of the above tests must be carried out by the testing team. However, it is not always possible to test the grammar, punctuation, syntax or art style and context of the reference localization, since it is quite difficult to find testers with knowledge of each of the languages into which the game is translated. The execution of these tests lies on the side of the localizers. Testers checking violations of the regional requirements of the tested localization, check for the lack of the necessary technical data, and identify defects in the display of text in game interfaces. (For more details, see section “7.3.3. Localization Testing Types”).

The preliminary stage of localization testing

This stage includes

- Providing testers with all necessary product documentation,
- Creation of a glossary and translation memory to help testers correctly interpret the terms used,
- Providing a previous version of the product, if it has already been localized earlier, for evaluation purposes,
- Selecting and configuring defect management tools - a document or a platform where all defects found during localization testing will be recorded.

Testing regional and cultural characteristics

This is one of the most important steps in localization testing. Screenshots or a localized build of the game will be used. The following needs to be tested:

- Date and time format for the selected region,
- Formats of phone numbers and addresses,
- Color schemes,
- Compliance of product names with regional standards,
- Currency format,
- Units.

Linguistic test

Language features are tested. The tester needs to make sure that:

- The same terminology is used,
- There are no grammatical defects,
- There are no spelling defects,
- Punctuation rules are followed,
- Correct text direction is used (right to left or left to right),
- The correct names of brands, cities, places, positions etc. are set.

User interface (or appearance)

It is important to make sure of the following:

- All captions in the pictures are localized,
- The layout of the localized version is the same as the original,
- Line breaks on pages / screens are placed in accordance with the rules of the target language,
- Conversations, pop-ups and notifications are displayed correctly,
- The length of the lines does not exceed the existing limits, and the text is displayed correctly (sometimes the translation text is longer than the original and does not fit on the buttons).

Functionality

It is necessary to test if the localized application is working correctly, so attention is drawn to:

- Functionality of a localized product,
- Functions for entering information,
- Support for special characters for different locales and languages,
- Support for keyboard shortcuts,
- Support for various fonts,
- Support for various format separators.

Video and scale testing includes testing:

- Correspondence of the length of the scale and the element of the game to which it belongs,
- Correspondence of the scale to the characters by gender,
- The purity of the sound (that is, the absence of interference, as well as the synchronization of the scale of the original and the translation and their loudness in relation to each other),
- The reliability of the soundtrack, including the historical aspect,
- Stylistic and cultural features of the sound (accents, speech features).

7.3.3 Localization Testing Types

"Box" localization

If a game is released and sold on physical media, what is written on the packaging is localized. If it is sold not on a physical medium, but on a platform, then its page in the store is translated: description and screenshots. Box localization is limited to this.

Interface localization

The description and the box will be translated in the game, the interface, the help page, the button labels, etc. An unusual type of localization arises in a situation when the inscription on the "Play" button is made in one language while the plot is completely in another language.

Text localization

Usually, all texts in the game are to be translated down to subtitles. This means that a user can listen and try to understand, for example, the African American slang in the game, but still see the subtitles in other language.

Localization with voice acting

Speech and dialogues are translated and voiced by the actors. If the localization with voice acting is done at a good level, it is not perceived as something foreign.

Graphic localization

Any game contains some kind of engine, design, graphic objects, textures - everything that is not text, for example, an inscription on a fence in a game. Graphic localization implies that all inscriptions inside must be translated. These can include, for example, newspapers, shop signs, and some notes.

Often the actions in the game take place in certain locations. At the same time, the approach to localization can be different. If some object (e.g., newspaper notes) helps in the plot, they must be translated, otherwise an important point will be lost. At the same time, the inscriptions on the walls in the language of the place where the game takes place do not need to be translated if they are just an arrangement.

Deep localization - cultural adaptation

This is a cultural adaptation, when the game is completely redone. All that remains is the source code and mechanics. Developers can remake textures, plot, dialogues, and character models and make a completely different game on the video game engine. This is done quite rarely, but this method of localization still occurs. This is done in cases where a game without such adaptation is not able to be used by the audience and sold in a particular market.

7.3.4 Areas of Responsibility of the Persons involved

Role	Responsibilities
Writer / Narrative Designer	• Development of a localization strategy and plan. • Definition of terminology, translation of concepts specific to the game genre and hard-to-translate terms related to the project. • Explaining the context to editors and translators.
Translator	• Text translation
Editor	• Proofreading of the material translated. • Testing the text for compliance with cultural characteristics together with moral and ethical aspects of the language in which localization is carried out. • Control of the uniformity of the translation style.
Localization tester	• Testing the correctness of the translation (context and meaning of remarks, coherence of the text, compliance with the style of play).

Role	Responsibilities
	• Testing the technical compliance of the translation (appearance, interfaces, functionality of fonts, separators, special characters, etc.). • Control of technical integration of translation (with internal and external applications).

7.4 Tools Support for Localization Testing

Localization testing tools can be used at different stages of testing and to address different issues. These tools include:

- Visual string comparison tools. These tools help to compare strings of reference and target localization.
- Automatic string comparison tools. Tools of this type include any auxiliary scripts, programs, utilities, as a result of which it is possible to obtain data on the presence of defects of any kind in the tested localization.
- Tools for testing the localization file structure for missing or redundant files.
- Examples of automated tools for testing localization include:
    - tools that compare screenshots for two localizations (reference and verified) and generate failures if there is a significant discrepancy,
    - tools that determine the presence of changes in the target localization based on the presence of changes in the reference,
    - tools that compare variables and numeric values in each pair of strings reference – and target localization.

8. References

8.1 Standards

[ISO25000] ISO/IEC 25000:2014, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE

[ISO25010] ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models

8.2 ISTQB documents

[ISTQB_AL_SEC] ISTQB Advanced Level Security Testing Syllabus, Version 2016

[ISTQB_ALTA_SYL] ISTQB Advanced Level Test Analyst Syllabus, Version 3.1.2

[ISTQB_ALTTA_SYL] ISTQB Advanced Level Technical Test Analyst Syllabus, Version 4.0

[ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 2012

[ISTQB_CTFL_MAT] ISTQB Mobile Application Tester, Version 2019

[ISTQB_EXAM_S&R] ISTQB Exam Structure and Rules, Game Testing, Version 1.0

[ISTQB_FL_AT] ISTQB Foundation Level Agile Tester Syllabus, Version 2014

[ISTQB_FL_PT] ISTQB Foundation Level Performance Testing Syllabus, Version 2018

[ISTQB_FL_SYL] ISTQB Foundation Level (Core) Syllabus, Version 2018

[ISTQB_GLOSSARY] ISTQB Glossary of Terms used in Software Testing, https://glossary.istqb.org/

[ISTQB_UT_SYL] ISTQB Foundation Level Usability Testing Syllabus, Version 2018

8.3 Books

[Nystrom14] Nystrom, R. (2014). Game programming patterns. Genever Benning., ISBN: 978-0990582908. URL: https://www.gameprogrammingpatterns.com/

[Gregory18] Gregory, J. (2018). Game engine architecture. AK Peters/CRC Press., ISBN: 978-1466560017. URL: https://www.gameenginebook.com/

[Buttfield19] Buttfield-Addison, P., Manning, J., & Nugent, T. (2019). Unity game development cookbook: essentials for every game. O'Reilly Media., ISBN: 9781491999158

[Lee16] Lee, J. (2016). Learning unreal engine game development. Packt Publishing Ltd., ISBN: 9781784398156

[Tavakkoli18] Tavakkoli, A. (2018). Game Development and Simulation with Unreal Technology. CRC Press., ISBN-13: 978-1498706247

[Romero19] Romero, M., & Sewell, B. (2019). Blueprints Visual Scripting for Unreal Engine: The faster way to build games using UE4 Blueprints. Packt Publishing Ltd., ISBN: 9781789347067

[Chandler11] Chandler, H. (2011). The Game Localization Handbook 2nd Edition, Jones & Bartlett Learning, ISBN: 0763795933

[Retsker81] Retsker, Ya. I. (1981). Textbook for translation from English into Russian. M.: Prosveschenie. (In Russian).

8.4 Links (Web/Internet)

Note: All references are current as of April 28, 2022.

- [ISTQB-Web] https://www.istqb.org/
- [URL1] https://research.ncl.ac.uk/game/mastersdegree/workshops/technicalrequirementschecklists/Technical%20Requirements%20Checklist%20Workshop.pdf
- [URL2] https://docs.microsoft.com/en-us/gaming/gdk/_content/gc/live/get-started/live-xbl-overview
- [URL3] https://gamepad-tester.com/
- [URL4] https://igda-website.s3.us-east-2.amazonaws.com/wp-content/uploads/2021/04/09142137/Best-Practices-for-Game-Localization-v22.pdf

9. Appendix A – Learning Objectives/Cognitive Level of Knowledge

The following learning objectives are defined as applying to this syllabus. Each topic in the syllabus will be examined according to the learning objective for it.

The learning objectives begin with an action verb corresponding to its cognitive level of knowledge as listed below.

Level 1: Remember (K1)

The candidate will remember, recognize and recall a term or concept.

Action verbs: Recall, recognize.

Examples
Recall the concepts of the test pyramid.
Recognize the typical objectives of testing.

Level 2: Understand (K2)

The candidate can select the reasons or explanations for statements related to the topic, and can summarize, compare, classify and give examples for the testing concept.

Action verbs: Classify, compare, differentiate, distinguish, explain, give examples, interpret, summarize

Examples	Notes
Classify test tools according to their purpose and the test activities they support.	
Compare the different test levels.	Can be used to look for similarities, differences or both.
Differentiate testing from debugging.	Looks for differences between concepts.
Distinguish between project and product risks.	Allows two (or more) concepts to be separately classified.
Explain the impact of context on the test process.	
Give examples of why testing is necessary.	
Infer the root cause of defects from a given profile of failures.	
Summarize the activities of the work product review process.	

Level 3: Apply (K3)

The candidate can carry out a procedure when confronted with a familiar task, or select the correct procedure and apply it to a given context.

Action verbs: Apply, implement, prepare, use

Examples	Notes
Apply boundary value analysis to derive test cases from given requirements.	Should refer to a procedure / technique / process etc.
Implement metrics collection methods to support technical and management requirements.	
Prepare installability tests for mobile apps.	
Use traceability to monitor test progress for completeness and consistency with the test objectives, test strategy, and test plan.	Could be used in a LO that wants the candidate to be able to use a technique or procedure. Similar to 'apply'.

10. Appendix B – Business Outcomes Traceability Matrix with Learning Objectives

This section lists the traceability between Certified Tester Game Testing Business Outcomes and Certified Tester Game Testing Learning Objectives.

Business Outcomes: Certified Tester Game Testing	GaMe-1	GaMe-2	GaMe-3	GaMe-4	GaMe-5	GaMe-6
GaMe-1 Understand the basic concepts of video games and game software testing	17					
GaMe-2 Determine risks, goals and game software requirements in accordance with the needs and expectations of stakeholders		12				
GaMe-3 Conceptually design, implement and execute basic game software tests			6			
GaMe-4 Know the approaches to game software testing and their purpose				9		
GaMe-5 Recognize the tools supporting game testing					7	
GaMe-6 Determine how testing activities align with the software lifecycle and reduce the cost of developing and publishing video games						6

			GaMe-1	GaMe-2	GaMe-3	GaMe-4	GaMe-5	GaMe-6
1. Specificity of Game Software Testing								
GaMe-1.1.1	K1	Recognize objectives and specifics of game testing	x					
GaMe-1.1.2	K2	Give examples of product risks in game software	x	x				
GaMe-1.1.3	K2	Give examples of specific defects related to game testing	x					
GaMe-1.1.4	K2	Summarize how the risks of game testing can be mitigated		x				
GaMe-1.1.5	K2	Compare the activities of game testing with those of playing	x					
GaMe-1.2.1	K1	Recognize specific roles and tasks in the game development team	x					
GaMe-1.3.1	K1	Recall testing activities throughout the game software lifecycle					x	
2. Testing Game Mechanics								
GaMe-2.1.1	K2	Classify the types of game mechanics	x					
GaMe-2.1.2	K2	Differentiate the testing of gameplay mechanics and non-gameplay mechanics			x			
GaMe-2.1.3	K2	Differentiate the testing of core mechanics and meta mechanics			x			
GaMe-2.1.4	K2	Differentiate the testing of client, server, and client-server mechanics			x			
GaMe-2.1.5	K2	Give examples of defects in game mechanics		x				
GaMe-2.2.1	K2	Summarize main approaches and test objects at different stages of creating a game product				x	x	
GaMe-2.2.2	K2	Distinguish the importance of testing game mechanics	x					
GaMe-2.2.3	K2	Distinguish the importance of reviewing documentation describing game mechanics	x					

GaMe-2.2.4	K3	Apply the fundamental approaches of testing game mechanics				x		
3. Graphics Testing								
GaMe-3.1.1	K2	Explain features of the graphic content of a game product	x					
GaMe-3.1.2	K2	Classify the types of defects in graphic content		x				
GaMe-3.2.1	K2	Summarize the main approaches to artistic testing				x		
GaMe-3.2.2	K2	Summarize the main approaches to technical testing				x		
GaMe-3.2.3	K2	Summarize main approaches to gameplay testing				x		
GaMe-3.3.1	K3	Apply the fundamental approaches of graphics testing			x			
GaMe-3.3.2	K2	Explain the importance of testing graphics for historical validity		x				
GaMe-3.4.1	K2	Summarize the usage of graphics testing tools					x	
4. Sound Testing								
GaMe-4.1.1	K1	Recall features of the sound content of a game product	x					
GaMe-4.2.1	K1	Recall types of defects in sound content		x				
GaMe-4.2.2	K2	Classify the defects in sound content		x				
GaMe-4.3.1	K2	Summarize the main approaches to content-auditory testing				x		
GaMe-4.3.2	K2	Summarize the main approaches for testing the mix of music and sounds				x		
GaMe-4.3.3	K2	Summarize the main approaches for testing music composition				x		
GaMe-4.4.1	K2	Explain the levels of testing audio-music content	x					
GaMe-4.4.2	K1	Recall the features of integrating sounds into the client	x					
GaMe-4.4.3	K1	Recall areas of responsibility of sound testing						x
GaMe-4.4.4	K3	Apply approaches to sound testing			x			
GaMe-4.5.1	K2	Summarize the usage of sound testing tools					x	
5. Game Level Testing								
GaMe-5.1.1	K1	Recall the components of the game level	x					
GaMe-5.1.2	K2	Classify the defects typical of game levels		x				
GaMe-5.2.1	K2	Summarize the tests carried out at various stages of the creation of game levels				x		x
GaMe-5.2.2	K2	Compare the areas of responsibility of specialists who take part in testing the game level						x
GaMe-5.3.1	K2	Summarize the usage of tools for testing game levels					x	
6. Game Controllers Testing								
GaMe-6.1.1	K2	Classify typical input devices and specialized ones	x					
GaMe-6.1.2	K2	Give examples of different input devices in terms of their application		x				
GaMe-6.1.3	K1	Recall different types of game controllers	x					
GaMe-6.1.4	K2	Classify the defects in a game product related to the specifics of game controllers, and possible causes of their occurrence		x				

GaMe-6.2.1	K2	Give examples for test conditions to be covered when testing game controllers		x				
GaMe-6.2.2	K2	Classify tasks for UX specialists, testers, and game designers during game testing						x
GaMe-6.3.1	K2	Summarize the usage of tools for testing the behavior of game controllers					x	
7. Localization Testing								
GaMe-7.1.1	K1	Recognize the main objectives of internationalization and localization	x					
GaMe-7.1.2	K1	Recall the localization test steps			x			
GaMe-7.1.3	K2	Compare internationalization and localization capabilities		x				
GaMe-7.2.1	K2	Classify the localization defects and their causes		x				
GaMe-7.3.1	K1	Recognize the full and partial localization testing				x		
GaMe-7.3.2	K3	Classify localization testing types						x
GaMe-7.3.3	K2	Summarize testing tasks for a writer, editor, translator, and localization tester						x
GaMe-7.4.1	K2	Summarize the usage of tools for testing of game localization					x	

11. Appendix C – Release Notes

This is the first release of the Certified Tester Game Testing syllabus. The development of this module had started in 2020 as the reaction on the growing market of the itnteractive entertainment software where testing is a highly technical field but there was no standard for game testing.

The Certified Tester Game Testing development team evaluates the number of specialists in the field of game development in more than 400.000.

Certified Tester Game Testing module gives opportunity to acquire an internationally recognized certification in order to be recognized among colleagues, employers, and customers.

12. Appendix D – Game Testing Specific and other Terms

Term Name	Definition
3D model	A 3D model of an object using visual mathematical forms.
accelerometer	A sensor that allows the software to determine and transmit information about how the video game controller is located in space at a given moment in time.
ambient	The sound of the environment that accompanies a certain location, situation, or stage of a video game.
animation	A technique for creating the illusion of moving images using a sequence of still images and replacing each other at a high frequency.
background sound	In-game background music to enhance a video game atmosphere.
binaural effect	An auditory illusion that is observed when oscillatory stimuli are delivered at two adjacent frequencies to each ear at the same time.
boss	A computer-controlled opponent that is much more difficult to defeat than a normal enemy in a video game.
client mechanics	Video game mechanics that work on the video game client side. See also video game mechanics.
collision	A technology responsible for the intersection of video game objects.
color coding	The process of marking objects with different colors as a means of identification.
computer-generated imagery (CGI)	The application of 3D computer graphics to create special effects.
concept stage	An initial planning stage of a game development project, which focuses on creating core concepts and writing initial design documents that describe the future video game.
core mechanics	Video game mechanics that define the intended game experience for the player. See also video game mechanics.
cultural adaptation	The process of adapting video game environment conditions to create the characteristic features of a given culture.
distortion	Change in the form of sound wave during processing. After The Oxford Companion to the English Language.
fighting	A genre of video games that simulate hand-to-hand combat of a small number of characters within a limited space and time.
first person shooter	A sub-genre of shooter video games centered on guns and other weapon-based combat in the first-person perspective, with the player experiencing the action through the eyes of the protagonist.
frame rate	The number of changed frames per unit of time.
gamepad	A type of two-handed game controller used with video game consoles. See also video game controller.
gameplay	The rules that describe the interaction between the video game world and the player.
game character	A person or any other entity acting in a game.
grey box	A stage in video game level development in which a "draft" 3D layout is created from one-color undefined forms to test game process

Term Name	Definition
gyroscope	A sensor that measures the orientation and angular velocity of a body in its rest frame
hacking	A dishonest way of gaining an advantage in a video game.
help	Documentation that explains the features of a program and helps the user understands its capabilities.
historical accuracy	Compliance of in-game content to its corresponding content in real life.
hit box. Synonym: affected area	A simplified 3D object model for object collision detection and handling within a video game.
inaccessible place	A location in the video game world that is incorrectly made available to a player.
lag	A delay in video game performance due to a slow Internet connection or memory leakage or severe CPU/RAM/HDD usage (e.g. windows/antivirus update in the background.
level design	A stage in video game development that involves creating video game levels and missions.
level editor	Software that is used to create and edit locations and environments within a video game.
level of detail (LoD)	A technology in which a simplified copy of an object is created to display the same object as viewed at a different distance.
level prototype	An early sample of a video game level that is created as a proof of concept. See also video game level.
locale	A set of options that define the language, region and any special variant preferences in the user interface of a video game.
loot box	An award in the form of virtual object(s) for completing a match, gaining an experience level, or other in-game achievement. After Wikipedia
mapping	A discipline in video game development for creating a level in a video game. See also video game level.
meta mechanics	Video game mechanics that define how a player is intended to use a video game. See also video game mechanics.
mobile object (Mob)	An enemy that roams a specific area of a video game.
morphing	A visual effect that creates the impression of a seamless transformation of one object into another.
multiplatform	The ability to use software on different platforms.
narrative	A story conveyed through video game mechanics. See also video game design.
non-player character (NPC)	A computer-controlled character. See also Mob.
occlusion	The process that imitates how sounds are perceived in "closed" environments.
platformer	A genre of video games that involves jumping on platforms, climbing stairs, and collecting items necessary to defeat enemies or complete a level.
player character	A character in video games, who is controlled by a player, typically a protagonist of the game's plot. See also Game character.
player versus player (PvP)	A video game mode in which a player interacts with one or more players.

Term Name	Definition
polygon	A set of vertices, edges, and faces that define the shape of a polyhedral object in 3D computer graphics and volume modeling.
post-production stage	The process of maintaining and distributing a video game, and marketing after its release.
pre-production stage	The process of planning and documenting the elements of a video game.
production stage	The process of developing a video game.
publisher	A company that finances, distributes and markets a video game. After Wikipedia
quest	A genre of video games that includes objective-based activities for either an interactive story or character level advancement. After Wikipedia
racing wheel	A type of two-handed video game controller used to imitate a steering wheel, pedals and gear lever.
raster	A type of graphic that represents a grid of pixels viewable on a screen.
reverb	An electronically produced echo effect. After Merriam-Webster
rigging	The process of building a model skeleton in a video game to animate it afterwards.
role-playing game (RPG)	A genre of video games in which a player assumes the roles of characters in a fictional setting.
scene lighting	The amount, size, color, and harshness of light surrounding a character to match the scene of a video game.
server mechanics	Game mechanics that work on the video game server. See also video game mechanics.
setting	The video game environment in which the action takes place.
skinning	The process of binding a 3D character to a model skeleton within a video game. See also rigging.
skipping	A faulty audio background that sounds like parts of the soundtrack are skipped.
sound discontinuity	An unexpected prolonged lack of sound or a sharp cutoff of sound.
sound effect	A sound other than speech or music made artificially for a video game object.
sound zone	A technology that allows the creation of in-game areas with real-life sound properties.
speedrunner	An attempt to complete a video game as fast as possible. After Wikipedia
stick	A video game controller whose mobility is limited to two degrees of freedom. See also video game controller.
store	A service that provides digital distribution of video games and other game content.
structural geometry	The technology that provides the terrain relief and the surface on which the video game characters can move.
terrain	A collection of elements related to the playing surface on which a character in a video game moves.
texture	A raster image superimposed on the surface of a polygonal model to give it color or imitate relief.

Term Name	Definition
tile	A small object or fragment from which a level or other large image is built in a video game.
touchscreen	A video game controller that relies on physical touch of a screen.
trackball	A game controller in which a freely-rotating ball is used as an input device for a video game. See also video game controller.
trigger	One or more actions that initiate an event.
video game	An electronic game in which a player controls images on a video screen. Merriam-Webster
video game console	An electronic device designed for video games
video game controller	A device used to provide input to a video game.
video game design	The process of creating the form and content of the gameplay of the video game being developed.
video game design document	Documentation containing a detailed description of the video game being developed.
video game designer	The person responsible for developing the rules and content of the gameplay. See also video game design.
video game engine	The software in which all other components of a video game are built.
video game environment	The content of the game: gameplay, game level, game objects, and in some cases, the game story.
video game level	A separate location within the virtual world of a video game.
video game mechanics	The actions within a video game defined by certain rules.
video game object	Any object that a player can interact with in a video game.
video game resource	A video game entity or characteristic important to a player to achieve the video game goals.
video game state	The value of all parameters and variables that describe all objects in a video game at a particular moment in time.
visual effect (VFX)	The creation or manipulation of any on-screen imagery that does not physically exist in real life.
volume	The degree of loudness or the intensity of a sound. Merriam-Webster
white box	A stage in video game level development in which a 3D layout is created from temporary models to test level geometry

13. Appendix E – Index