**Pac-man game using JavaFx**

**Bonafide Certificate**

This work is done by S.Koushik *(192111699)* from the Computer Science and Engineering Department during Slot A Programming in Java(CSA0915) in the duration from December 2023 to March 2024 under the guidance of T.Vincent Gnanaraj

**Abstract**

Delve into the maze of excitement with Pac-Man JavaFX rendition, offering thrilling chases and strategic escapes. Navigate through labyrinthine pathways, gobbling up pellets while evading relentless ghosts. Engage in a classic arcade adventure reimagined with vibrant graphics and dynamic animations. Immerse yourself in nostalgic gameplay enhanced by modern JavaFX functionalities. Experience the pulse-pounding action as you strive to conquer each level and achieve high scores. Get ready to embark on a journey filled with nostalgia, challenge, and endless fun.

**Keywords:** Pac-Man, JavaFX, Game development, Maze, Ghosts

1. **Introduction**

Pac-Man is a classic arcade game where players control a yellow character navigating through a maze, eating pellets, and avoiding ghosts. With its simple yet addictive gameplay, Pac-Man has remained a beloved favorite since its release in 1980. Now, with the advent of JavaFX, developers can bring the nostalgia of Pac-Man to modern platforms, offering players a chance to relive the excitement of this timeless game.

Despite its enduring popularity, one common problem with the current Pac-Man game is its lack of customization options. Players often find themselves limited to the standard maze layout and ghost behavior, leading to a repetitive gameplay experience. Additionally, the game may lack certain features that could enhance replayability and engagement, such as different difficulty levels, power-ups, or bonus stages.

To address these issues, developers can leverage JavaFX's flexibility and versatility to create a more dynamic and customizable Pac-Man experience. By incorporating features such as customizable maze layouts, varied ghost behaviors, and power-ups, developers can offer players a more personalized and challenging gameplay experience. Additionally, adding new game modes or bonus stages can further enhance the replay value of the game, keeping players engaged for hours on end.

1. **Existing system**

In the existing Pac-Man game implemented using JavaFX, the system operates by rendering the game's graphical elements on a canvas using JavaFX's scene graph. User input, such as keyboard commands, is captured and translated into game actions, such as moving the Pac-Man character. The game logic manages the movement of Pac-Man and the ghosts, collision detection, scoring, and level progression. JavaFX's animation capabilities are utilized to create smooth movement and transitions between game states, enhancing the player's experience.

However, despite its functionality, the existing system in the Pac-Man game using JavaFX faces several challenges. One issue is the lack of efficient collision detection algorithms, leading to occasional inaccuracies or glitches when Pac-Man interacts with the maze walls, pellets, or ghosts. Additionally, the system may struggle to handle complex game logic efficiently, particularly as the game's complexity increases with higher levels or additional features. Furthermore, maintaining code cleanliness and modularity can become challenging as the project grows, potentially hindering future development and updates.

1. **Proposed system**

In the proposed system for the Pac-Man game using JavaFX, our goal is to enhance the accuracy of gameplay mechanics, such as movement and collision detection. By leveraging JavaFX's capabilities, we plan to implement algorithms that optimize Pac-Man's movement patterns and improve the precision of collision detection between Pac-Man, ghosts, and other game elements. This increased accuracy will result in a more seamless and enjoyable gaming experience for players, as they navigate through the maze and evade the ghosts with greater precision.

The novelty of this research lies in its application of JavaFX for enhancing the classic Pac-Man game. While Pac-Man has been a popular and enduring title since its inception, there is still room for innovation and improvement in its gameplay mechanics. By utilizing JavaFX's modern graphics and user interface features, we aim to breathe new life into the game while staying true to its nostalgic charm. This project also provides an opportunity to explore the intersection of classic game design principles and modern software development techniques, paving the way for future advancements in game development.

1. **Requirement specification**

**Hardware Requirements**

**CPU:** Intel(R) Core(TM) i5-1035G4 CPU @ 1.10GHz 1.50 GHz

**RAM:**16GB

**SSD:**512GB

**KEYBOARD:** Membrane

**MOUSE:** Optical

**MONITOR:** LED

**INTERNET:** Wifi

**Software Requirements**

**DEVELOPMENT SDK:** Java JDK(21)

**RUNTIME ENVIRONMENT:** JRE-8

**OPERATING SYSTEM:** Windows 11 64-bit Processor

**PACKAGES:** JavaFx

**Human Requirements**

**TEAM SIZE:**1

**SUPERVISOR:**1

**TOTAL:**2

**FINANCIAL REQUIREMENTS**

**BUDGET:** 1000 to 15000 depending on implementing and quality metrics

1. **Methodology**

**SDLC MODEL:** The Software Development Life Cycle (SDLC) is a structured methodology used to plan, design, develop, test, deploy, and maintain software systems. It provides a systematic approach for managing the entire software development process from start to finish, ensuring that projects are completed on time, within budget, and with high quality. The SDLC typically consists of several phases, such as requirements gathering, design, implementation, testing, deployment, and maintenance, with each phase building upon the previous one. By following the SDLC model, development teams can effectively manage and control the software development process, resulting in the delivery of successful and reliable software solutions.

**SDLC MODEL FOR Pac-man game**

* Define the objectives of the Pac-Man game, including its features, mechanics, and overall design. This phase involves brainstorming ideas, creating a project plan, and establishing milestones.
* Develop the visual and technical aspects of the game, such as character designs, level layouts, game mechanics, and user interface elements. This includes creating wireframes, storyboards, and mock-ups to visualize the game's structure and flow.
* Implement the Pac-Man game according to the design specifications. This involves writing code, creating art assets, integrating audio, and testing the game's functionality at various stages of development.
* Conduct thorough testing to identify and resolve bugs, glitches, and gameplay issues. This includes unit testing, integration testing, regression testing, and user acceptance testing to ensure the game meets quality standards and player expectations.
* Release the Pac-Man game to the intended platform(s), such as mobile devices, gaming consoles, or web browsers. This may involve submitting the game to app stores, distributing physical copies, or hosting the game on online platforms. Additionally, post-launch support and updates may be provided to address any issues or add new features based on player feedback.

1. **Analysis**

**Data analysis**

* Analyze metrics such as total score, level progression, and number of lives remaining to understand how well players are performing in the game. This could include tracking average score per level, time taken to complete each level, and frequency of game overs.
* Study the movement patterns and interactions of the ghosts in the game. This could involve analyzing their paths, speed, and decision-making processes to determine how challenging they are for players to avoid.
* Examine how often players utilize power-ups such as power pellets and fruits, as well as their impact on gameplay. This could include tracking the duration of invincibility after consuming a power pellet and the number of ghosts eaten during this time.
* Analyze player movements and decision-making strategies during gameplay. This could involve studying patterns such as preferred routes through the maze, timing of power-up consumption, and avoidance tactics when confronted by ghosts.
* Evaluate the overall balance of the game by considering factors such as difficulty progression, distribution of power-ups and enemies, and fairness of scoring mechanics. This could involve adjusting game parameters to ensure a satisfying experience for players of varying skill levels.

**Software analysis**

* The software implements the classic Pac-Man gameplay mechanics, where the player controls the character to navigate through a maze, eating pellets and avoiding ghosts. The game mechanics should be designed to be intuitive and responsive, providing an enjoyable user experience.
* The software includes graphics and animation to render the game environment, including the maze, Pac-Man character, ghosts, pellets, and power-ups. Smooth animation and vibrant visuals enhance the gaming experience and contribute to the overall atmosphere of the game.
* The software incorporates AI algorithms to control the behavior of the ghosts, which adds complexity and challenge to the game. The ghosts should exhibit various strategies, such as chasing Pac-Man, patrolling specific areas, or fleeing when Pac-Man consumes a power-up.
* The software includes multiple levels with increasing difficulty, offering a variety of maze layouts and challenges for the player to overcome. Level design plays a crucial role in keeping the game engaging and encouraging players to continue playing to reach higher levels.
* The software tracks the player's score and provides feedback through visual and auditory cues, such as displaying the score on the screen, playing sound effects when Pac-Man eats pellets or power-ups, and showing animations when Pac-Man is caught by a ghost or completes a level. Feedback mechanisms help to motivate players and reinforce their actions in the game.

**Hardware analysis**

* The hardware needs to have a capable processor to handle the game logic, enemy AI, and player inputs in real time. A modern processor with sufficient processing power, such as a quad-core CPU, would be suitable for running a Pac-Man game smoothly.
* While Pac-Man doesn't require cutting-edge graphics, it still needs a graphics processor capable of rendering the game's visuals smoothly. A dedicated graphics card or integrated graphics with support for 2D graphics acceleration would suffice.
* Sufficient RAM is essential for storing game assets, such as sprites, textures, and audio files, as well as running the game code. At least 4GB of RAM would be necessary for smooth gameplay, although more would be beneficial for multitasking.
* Pac-Man itself doesn't require much storage space, but the hardware should have enough storage for the operating system, game files, and any additional software or media. A solid-state drive (SSD) would provide fast loading times and responsiveness.
* The hardware should support various input devices, such as keyboards, gamepads, or touchscreens, depending on the platform. Additionally, responsive input processing is crucial for accurate control in a fast-paced game like Pac-Man.

**MAINTENANCE**: The software is under good Maintenance.

1. **Human Resources**

We have gained enough experience in Java Programming to complete this Project. The Supervisor is capable of training this project.

1. **External Resources**

On-site training, Internships, Workshops, and Funded Projects could be very helpful resources for this project.

1. **Design**

For designing this project, we have developed an architecture diagram, Sequence diagram, and ER diagram

**FRONTEND DESIGN:** Using Fonts, Layout and GUI(Graphical User Interface)

**BACKEND DESIGN:** No Backend Required for this project

**Architecture diagram**

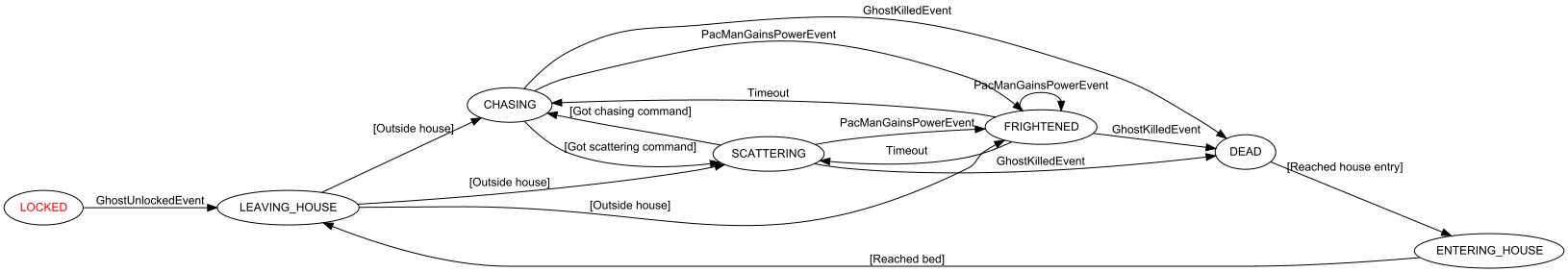


Figure 1: Architecture diagram of the game starting from leaving the house to entering the house of the game.

**ER diagram**

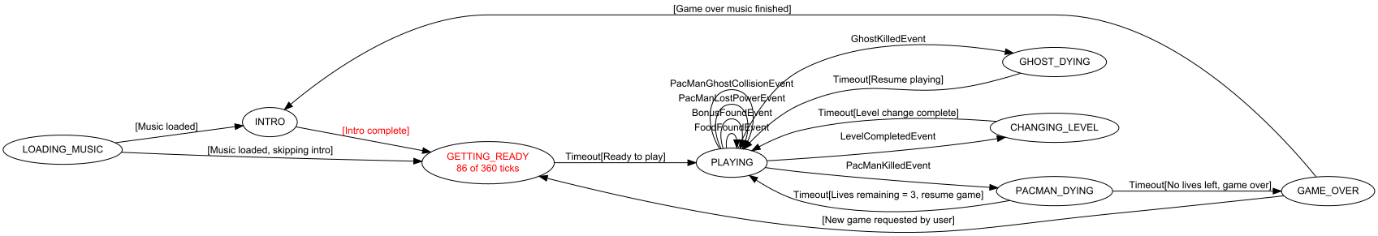
****

Figure 2: The ER diagram is a visual representation of the data model that shows the relationships between entities stored in a database.

**Sequence diagram**

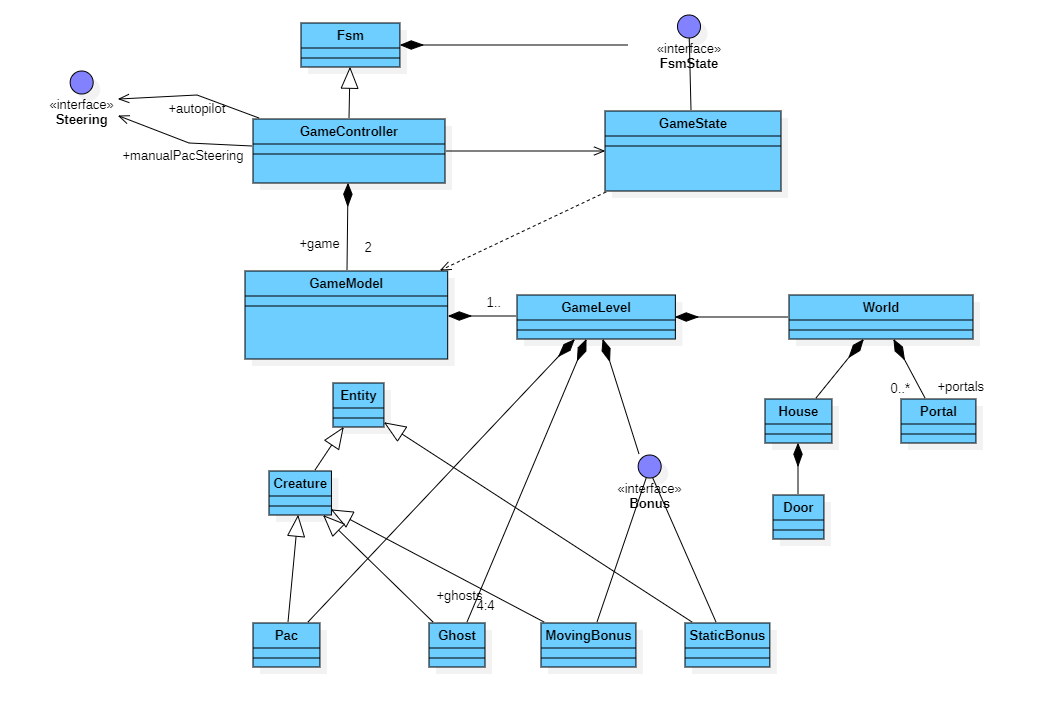
****

Figure 3 Sequence diagram using UMI software.

1. **Coding**

**Syntax**

1. public class MyClass { } – This is a class declaration. MyClass is the name of the class.
2. public static void main(String[] args) { } – This is the main method that gets executed when you run your Java program.
3. int myVar = 5; – This is a variable declaration. myVar is a variable of type int and is assigned the value 5.
4. final double PI = 3.14; – This is a constant declaration. PI is a constant of type double and is assigned the value 3.14.
5. if (condition) { } – This is an if statement. The code inside the braces {} is executed if the condition is true.
6. if (condition) {} else {} – This is an if-else statement. If the condition is true, the code in the first block is executed; otherwise, the code in the else block is executed.
7. switch (variable) {case value: break; default: break;} – This is a switch statement. It allows a variable to be tested for equality against a list of values.
8. for (initialization; condition; increment) {} – This is a for loop. It is used to repeatedly execute a block of code until a certain condition is met.
9. while (condition) {} – This is a while loop. It repeatedly executes a block of code as long as a certain condition is true.
10. do {} while (condition); – This is a do-while loop. It is similar to a while loop, but the condition is tested after the execution of the block of code.
11. int[] myArray = new int[10]; – This is an array declaration. myArray is an array of int type with a size of 10.
12. public returnType functionName(parameters) {} – This is a function declaration. functionName is the name of the function, returnType is the data type of the value the function returns, and parameters are input to the function.
13. try {} catch (ExceptionType e) {} – This is a try-catch block. It is used to handle exceptions and errors that occur in a block of code.
14. MyClass obj = new MyClass(); – This is how to create an object. obj is an object of the class MyClass.
15. obj.memberName; – This is how to access an object’s members. memberName is the name of a member (variable or method) of the object obj.

**source code**

package de.amr.games.pacman.test;

import de.amr.games.pacman.controller.GameController;

import de.amr.games.pacman.lib.Vector2f;

import de.amr.games.pacman.lib.Vector2i;

import de.amr.games.pacman.model.GameModel;

import de.amr.games.pacman.model.GameVariant;

import de.amr.games.pacman.model.actors.StaticBonus;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import static org.junit.Assert.\*;

/\*\*

\* @author Armin Reichert

\*/

public class PacManGameTest {

private GameModel game;

@BeforeClass

public static void setUp() {

GameController.create(GameVariant.PACMAN);

}

@Before

public void setUpTest() {

game = GameController.it().game();

game.reset();

game.createAndStartLevel(1);

}

@Test

public void testGameControllerCreated() {

assertNotNull(GameController.it());

}

@Test(expected = IllegalStateException.class)

public void testGameControllerCreatedTwice() {

GameController.create(GameVariant.MS\_PACMAN);

}

@Test

public void testLevelInitialized() {

assertTrue(game.level().isPresent());

var level = game.level().get();

assertEquals(1, level.number());

assertEquals(0, level.numGhostsKilledInLevel());

assertEquals(0, level.numGhostsKilledByEnergizer());

assertEquals(0, level.cruiseElroyState());

}

@Test

public void testPacCreatedAndInitialized() {

game.level().ifPresent(level -> {

var pac = level.pac();

assertEquals(0, pac.restingTicks());

assertEquals(0, pac.starvingTicks());

});

}

@Test

public void testGhostsCreatedAndInitialized() {

game.level().ifPresent(level -> {

var redGhost = level.ghost(GameModel.RED\_GHOST);

assertEquals(-1, redGhost.killedIndex());

assertNotEquals(Vector2f.ZERO, level.ghostRevivalPosition(redGhost.id()));

assertNotEquals(Vector2i.ZERO, level.ghostScatterTarget(redGhost.id()));

var pinkGhost = level.ghost(GameModel.PINK\_GHOST);

assertEquals(-1, pinkGhost.killedIndex());

assertNotEquals(Vector2f.ZERO, level.ghostRevivalPosition(pinkGhost.id()));

assertNotEquals(Vector2i.ZERO, level.ghostScatterTarget(pinkGhost.id()));

var cyanGhost = level.ghost(GameModel.CYAN\_GHOST);

assertEquals(-1, cyanGhost.killedIndex());

1. **Testing**

**MODULE TESTING**

* Verify that all elements of the Pacman game's GUI, including buttons, labels, and images, are displayed correctly and respond appropriately to user interactions such as clicks and keyboard inputs.
* Test the functionality of various game mechanics such as movement of Pacman and ghosts, collision detection, score calculation, power-ups, and level transitions to ensure that they work as intended and provide a smooth gaming experience.

**BLACKBOX TESTING**

* Verify that all game mechanics function as expected, such as Pacman movement, ghost behavior, pellet consumption, level transitions, and scoring.
* Validate the responsiveness and robustness of user inputs, including keyboard controls for Pacman movement and any menu interactions.

**WHITEBOX** **TESTING**

Whitebox testing for a Pacman game using JavaFX might involve:

* This includes testing the movement of Pacman, the behavior of ghosts, collision detection with walls and other game elements, and the scoring system. Test cases would be designed to cover all possible scenarios, including edge cases and boundary conditions.
* This involves verifying that all graphical elements, such as buttons, menus, and score displays, are functioning correctly and are properly integrated with the game logic. Test cases would focus on user interactions and ensuring that the interface responds appropriately to user input.

**INTEGRATION TESTING**

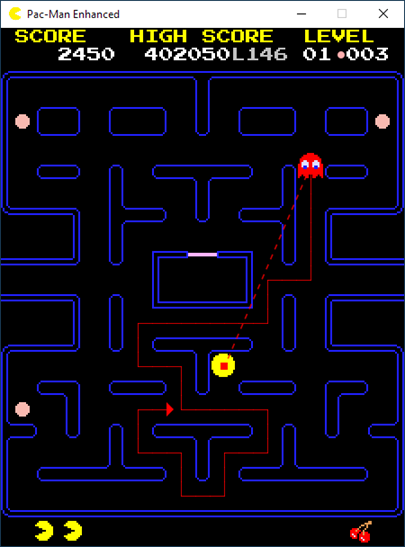
* Integrate and test the interaction between Pacman character movement and collision detection with walls and other game elements.
* Verify the integration of Pacman's score tracking and power-up mechanics with the overall game functionality in the JavaFX environment.

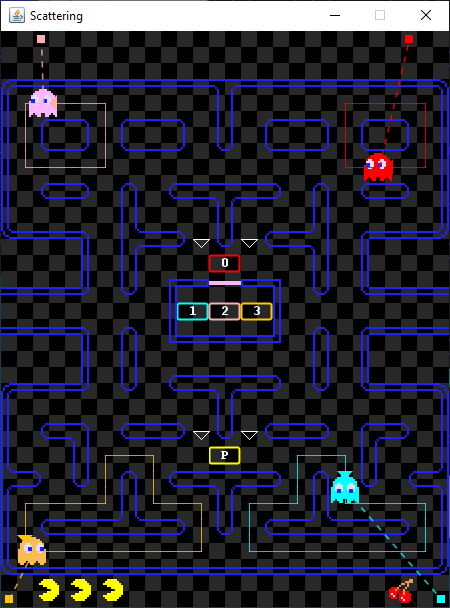
**SYSTEM TESTING**

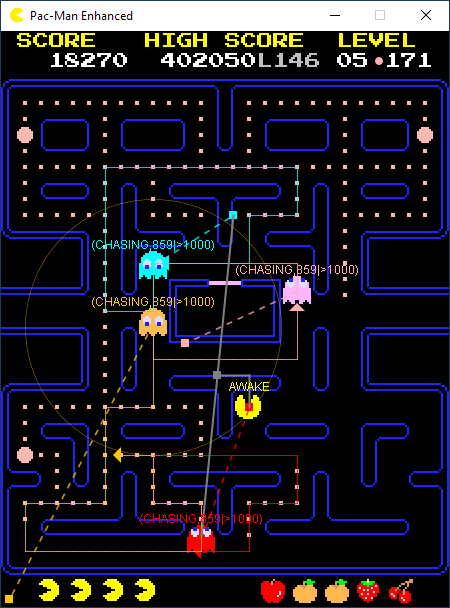
* Conduct comprehensive testing of Pacman game functionalities, including movement, collisions, power-ups, score tracking, and level progression.
* Verify compatibility and smooth operation across different platforms and screen resolutions, ensuring a seamless gaming experience for users.

1. **Implementation**

To implement a Pacman game concisely using JavaFX, you would first need to set up a graphical user interface (GUI) using JavaFX's Scene and Stage classes. Then, you'll create classes to represent Pacman, ghosts, and the game board. Implement movement mechanics for Pacman and ghosts using event handling, and ensure collision detection with walls and other game elements. Use JavaFX's animation capabilities to update the game state and render it on the GUI continuously. Add scoring, level progression, and game-over conditions to make the game engaging. Finally, incorporate sound effects and visual feedback to enhance the player experience.







1. **Maintenance**

This project requires maintenance according to software updates and hardware changes

1. **Conclusion**

In conclusion, developing a Pacman game using JavaFX provides a dynamic and visually engaging experience for players. Through the use of JavaFX's graphics and animation capabilities, we can create a faithful adaptation of the classic arcade game while also adding our own unique features and enhancements. By leveraging JavaFX's event handling and user interface components, we can ensure smooth gameplay and intuitive controls. Additionally, JavaFX's cross-platform compatibility allows our Pacman game to run seamlessly on various devices and operating systems. With its robust tools and libraries, JavaFX enables us to implement challenging gameplay mechanics, intricate level designs, and vibrant graphics, making the Pacman experience truly immersive and enjoyable for players of all ages.

1. **Acknowledgment**

We thank Oracle for providing Java software. We thank our guide for Technical support and thank mentor for moral support and we also thank our principal for accommodation support

1. **Reference**

[1]Lima, H., Jr, F. C. de Andrade, A. Montenegro, and E. Clua. n.d. “An Evaluation of JavaFX as a 2D Game Creation Tool.” *Citeseer*.

[2]Leroy, D., E. Bousse, and M. Wimmer. 2017. “Create and Play Your Pac-Man Game with the GEMOC Studio (Tool Demonstration).” *Workshop on …*.

[3]Zavisca, Ernest, and Gary Beltowski. 1982. *Break a Million! At Pac-Man*. Delair.

[4]Baimagambetov, Almas. 2022. “AI Case Study: Develop a Maze Action Game.” In *Learn JavaFX Game and App Development: With FXGL 17*, edited by Almas Baimagambetov, 113–54. Berkeley, CA: Apress.

[5]Jackson, Wallace. 2017. “An Introduction to Game Design: Game Design Concepts, Genres, Engines, and Techniques.” In *Pro Java 9 Games Development: Leveraging the JavaFX APIs*, edited by Wallace Jackson, 73–86. Berkeley, CA: Apress.

[6]<https://github.com/jbaskauf/pacman>

[7]<https://www.reddit.com/r/JavaFX/comments/11a4n0o/pacmanjavafx_a_3d_2d_pacman_and_ms_pacman/>

[8]<https://zetcode.com/javagames/pacman/>