Gliwice
Akademic Year 2024/2025

Computer Programming 3

FINAL REPORT

C++ SFML GAME MAKE

OTTER ESCAPE

Instructor :
Ph D. inż. Piotr Fabian

Made by:
Hyunseok Cho 313615

Contents

1. **Introduction**

   This report presents a 2D graphic game developed as part of a hackathon project, with the theme "Escape the Loop." The game was created using C++ and the SFML (Simple and Fast Multimedia Library) to provide an engaging and interactive experience. The primary objective of the game is to navigate through various obstacles and challenges to escape a looping environment. This project was a collaborative effort with my team members during the hackathon, and it has been approved by the professor to be submitted as a final project for my programming course.

2. **Design and Architecture**

   The game is designed with a modular architecture, allowing for easy maintenance and scalability. The core components of the game include:

- **Game Loop: The main loop that controls the flow of the game, handling events, updating game states, and rendering graphics.**
- **Player: The player character is controlled by the user, with functionalities for movement, rotation, and fuel management. The player must navigate through the game environment to reach the goal.**
- **Animation: Manages the animations for various game objects, ensuring smooth transitions between frames.**
- **Physics: Provides utility functions for physical calculations, such as applying forces and calculating distances, to simulate realistic movements and interactions.**
- **Planets and Planet Manager: Represents obstacles in the game, with planets following predefined paths. The Planet Manager oversees the initialization and updating of these celestial bodies.**
- **UI: Displays essential information to the player, such as position, velocity, and remaining fuel, enhancing the user experience.**
- **Main Menu: Offers a user-friendly interface for starting the game, accessing information, and exiting the application.**

  The game leverages object-oriented programming principles to encapsulate functionalities within classes, promoting code reusability and clarity. Each component interacts seamlessly with others, creating a cohesive and dynamic gaming experience.

3. **Implementation details**

Animation.h & Animation.cpp

Animation.h

- **Member Variables**
  - **sf::Vector2u imageCount: Stores the number of images in the animation sheet.**
  - **sf::Vector2u currentImage: Stores the index of the currently displayed image.**
  - **float totalTime: Stores the elapsed time since the animation started.**
  - **float switchTime: Stores the time required to switch frames.**
  - **sf::IntRect uvRect: Stores the texture coordinates of the current frame.**
- **Constructors**
  - **Animation(sf::Texture* texture, sf::Vector2u imageCount, float switchTime): Initializes the animation with texture, image count, and frame switch time.**
  - **Animation(): Default constructor.**
- **Destructor**
  - **~Animation(): Destructor.**
- **Member Functions**
  - **void setParamters(sf::Texture* texture, sf::Vector2u imageCount, float switchTime): Sets the parameters of the animation. Called after using the default constructor.**
  - **void update(int row, float deltaTime): Updates the current frame of the animation using deltaTime to control frame switching.**

Animation.cpp

- **Connections and Integration**
  - **The Animation class is used in the GraphicalObject class. GraphicalObject contains an Animation object and updates the animation in its update function.**

Main.cpp

Main.cpp

- **main Function**
  - **int main(): Entry point of the program. Creates a Game object and calls its run method to start the game. Handles exceptions if they occur.**

Game.h & Game.cpp

Game.h

- **Member Variables**

- **sf::RenderWindow* mWindow: Represents the game window.**
- **sf::Font font, font_title: Stores the fonts used in the game.**
- **UI* ui: Manages the user interface.**
- **MainMenu* mainmenu: Manages the main menu.**
- **Player player: Represents the player character.**
- **TextureManager textureManager: Manages textures.**
- **PlanetManager planetsManager: Manages planets.**
- **sf::Clock clk: Clock for time measurement.**
- **float deltaTime: Stores the time difference between frames.**
- **sf::RectangleShape goal: Represents the goal area.**

- **Constructor**
  - **Game(): Initializes the game settings, creates the window, loads fonts, and initializes UI and main menu.**
- **Destructor**
  - **~Game(): Frees dynamically allocated memory.**
- **Member Functions**
  - **void run(): Executes the game loop.**
  - **void processEvents(): Handles user input.**
  - **void update(): Updates the game state.**
  - **void render(): Renders the current game state to the screen.**
  - **void renderMainMenu(): Renders the main menu.**
  - **void renderCredit(): Renders the credits screen.**
  - **void handleMainMenuEvents(sf::Event& event): Handles events in the main menu.**
  - **void handleCreditsEvents(sf::Event& event): Handles events in the credits screen.**
  - **void handleGameOverEvents(sf::Event& event): Handles events in the game over state.**
  - **void handleWinEvents(sf::Event& event): Handles events in the win state.**
  - **void renderGameOver(): Renders the game over screen.**
  - **void renderWin(): Renders the win screen.**

Game.cpp

- **Connections and Integration**
  - **The Game class interacts with Player, UI, MainMenu, TextureManager, and PlanetManager. It runs the game loop and performs appropriate updates and rendering based on the current state.**

GraphicalObject.h & GraphicalObject.cpp

GraphicalObject.h

- **Member Variables**
  - **sf::RectangleShape rectangle: Represents the shape of the graphical object.**
  - **Animation animation: Manages the animation of the object.**
- **Constructor**
  - **GraphicalObject(): Default constructor.**
- **Destructor**
  - **~GraphicalObject(): Destructor.**
- **Member Functions**
  - **void update(float deltaTime): Updates the physical state and animation of the object.**
  - **void render(sf::RenderWindow* win, float deltaTime): Renders the object to the screen.**
  - **void setPositionExternal(sf::Vector2f newPos): Sets the position of the object.**
  - **void setSizeMidOriginRect(sf::Vector2f newSize): Sets the size and origin of the object.**
  - **void prepAnimParams(sf::Texture* t, sf::Vector2u imageCount, float switchTime): Sets the animation parameters.**

GraphicalObject.cpp

- **Connections and Integration**
  - **GraphicalObject inherits from PhysicalObject to have physical properties and uses an Animation object to manage animations.**

MainMenu.h & MainMenu.cpp

MainMenu.h

- **Member Variables**
  - **sf::Text titleText, playButton, infoButton, exitButton, madeby: Text elements of the menu.**
  - **sf::Texture background: Stores the background image.**
  - **sf::Sprite background_sprite: Background sprite.**
  - **int selectedOption: Represents the currently selected menu option.**
- **Constructor**
  - **MainMenu(const sf::Font& font, const sf::Font& font_title): Initializes the menu settings.**
- **Member Functions**

- **void render(sf::RenderWindow& window) const: Renders the menu to the screen.**
- **void handleMouseInput(const sf::Vector2f& mousePos): Changes button colors based on mouse input.**
- **int getSelectedOption() const: Returns the selected option.**

MainMenu.cpp

- **Connections and Integration**
  - **MainMenu is used in the Game class and handles user input and rendering in the main menu state.**

PhysicalObject.h & PhysicalObject.cpp

PhysicalObject.h

- **Member Variables**
  - **sf::RectangleShape rectangle: Represents the shape of the object.**
  - **float gravityConst: Stores the gravitational constant.**
  - **sf::Vector2f pos, velocity: Stores the position and velocity of the object.**
  - **std::vector<sf::Vector2f> forces: Stores the forces acting on the object.**
- **Constructor**
  - **PhysicalObject(sf::Vector2f p = sf::Vector2f(0,0), sf::Vector2f v = sf::Vector2f(0,0)): Sets the initial position and velocity of the object.**
- **Destructor**
  - **~PhysicalObject(): Destructor.**
- **Member Functions**
  - **void applyForces(float deltaTime): Applies forces to the object.**
  - **void updatePosition(float deltaTime): Updates the position of the object.**
  - **void updatePhysics(float deltaTime): Updates the physical state of the object.**

PhysicalObject.cpp

- **Connections and Integration**
  - **PhysicalObject is inherited by GraphicalObject and Player classes to manage physical properties. It works with Physics class functions for physical calculations.**

Physics.h & Physics.cpp

Physics.h

- **Member Functions**
  - **static float distanceBetweenPoints(sf::Vector2f p1, sf::Vector2f p2): Calculates**

the distance between two points.

- static float vectorLength(sf::Vector2f v): Calculates the length of a vector.
- static sf::Vector2f makeVersor(sf::Vector2f v): Converts a vector to a unit vector.
- static float dotProduct(sf::Vector2f v, sf::Vector2f u): Calculates the dot product of two vectors.
- static void applyGravityForce(PhysicalObject& p1, PhysicalObject& p2): Calculates and applies gravitational force between two objects.
- static void applyThrust(PhysicalObject& p, float thrustValue, float angle): Applies thrust to an object.

Physics.cpp

- Connections and Integration
  - The Physics class is used in PhysicalObject and Player classes to perform physical calculations and control object movement.

Planet.h & Planet.cpp

Planet.h

- Member Variables
  - sf::Vector2f orbitCenter: Represents the center of the orbit.
  - float orbitTravellingTime: Stores the time taken to travel between discrete points of the orbit.
  - float velocityMag: Stores the magnitude of velocity.
  - std::vector<sf::Vector2f> path: Stores the path to follow along the orbit.
  - int destinationIndex: Stores the index of the current target point.
- Constructor
  - Planet(): Default constructor.
- Destructor
  - ~Planet(): Destructor.
- Member Functions
  - void createPath(sf::Vector2f origin, float radius, float intAngle): Creates the orbit path for the planet.
  - void followPath(float deltaTime): Follows the path along the orbit.
  - void updatePlanet(float deltaTime): Updates the state of the planet.

Planet.cpp

- Connections and Integration
  - The Planet class is managed by the PlanetManager and represents planets moving along an orbit.

PlanetManager.h & PlanetManager.cpp

PlanetManager.h

- **Member Variables**
  - **TextureManager* textureManag: Pointer to the texture manager.**
  - **std::vector<Planet> planets: Vector storing planet objects.**
- **Constructor**
  - **PlanetManager(): Default constructor.**
- **Destructor**
  - **~PlanetManager(): Destructor.**
- **Member Functions**
  - **void initPlanets(sf::RenderWindow* w): Initializes the planets.**
  - **void updatePlanetManager(float deltaTime): Updates the state of all planets.**
  - **void renderPlanets(sf::RenderWindow* w, float deltaTime): Renders the planets to the screen.**

PlanetManager.cpp

- **Connections and Integration**
  - **PlanetManager is used in the Game class to manage and update multiple planets.**

Player.h & Player.cpp

Player.h

- **Member Variables**
  - **sf::Vector2f position: Stores the position of the player.**
  - **float rotation: Stores the rotation angle of the player.**
  - **float fuel: Stores the fuel level of the player.**
- **Constructor**
  - **Player(float x, float y): Sets the initial position of the player.**
- **Member Functions**
  - **void handleInput(const sf::Keyboard& keyboard, float deltaTime): Handles keyboard input.**
  - **void update(float deltaTime): Updates the state of the player.**
  - **void render(sf::RenderWindow& window): Renders the player to the screen.**
  - **void accelerate(): Accelerates the player.**
  - **void rotateLeft(): Rotates the player to the left.**
  - **void rotateRight(): Rotates the player to the right.**
  - **sf::Vector2f getPosition() const: Returns the position of the player.**

- sf::Vector2f getVelocity() const: Returns the velocity of the player.
- float getFuel() const: Returns the fuel level of the player.
- bool isOutOfFuel() const: Returns whether the player is out of fuel.

Player.cpp

- Connections and Integration
  - The Player class is used in the Game class to manage the player's movement and state. It uses the Physics class for physical calculations.

TextureManager.h & TextureManager.cpp

TextureManager.h

- Member Variables
  - std::map<std::string, sf::Texture> manager: Map storing textures.
- Constructor
  - TextureManager(): Default constructor.
- Destructor
  - ~TextureManager(): Destructor.
- Member Functions
  - void addTexture(std::string name, std::string fileName): Adds a new texture.
  - sf::Texture& getTexture(std::string s): Returns a stored texture.

TextureManager.cpp

- Connections and Integration
  - TextureManager is used in Game, PlanetManager, and Player to manage textures.

UI.h & UI.cpp

UI.h

- Member Variables
  - sf::RectangleShape ui_background: Represents the background of the UI.
  - sf::Text position, velocity, fuel: Text displaying player information.
- Constructor
  - UI(const sf::Font& font): Initializes the UI settings.
- Member Functions
  - void update(const sf::Vector2f& position, const sf::Vector2f& velocity, float fuel): Updates player information.
  - void render(sf::RenderWindow& window) const: Renders the UI to the screen.

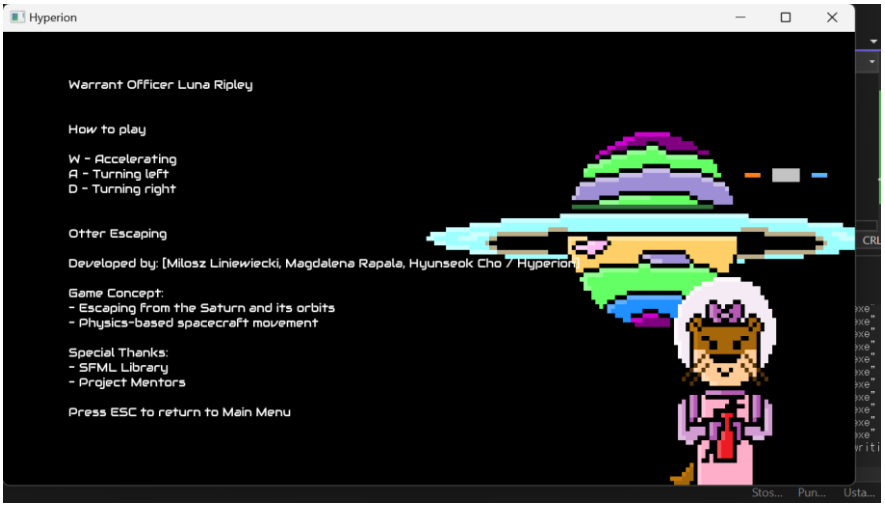
UI.cpp

- **Connections and Integration**
  - **The UI class is used in the Game class to display player state information on the screen.**

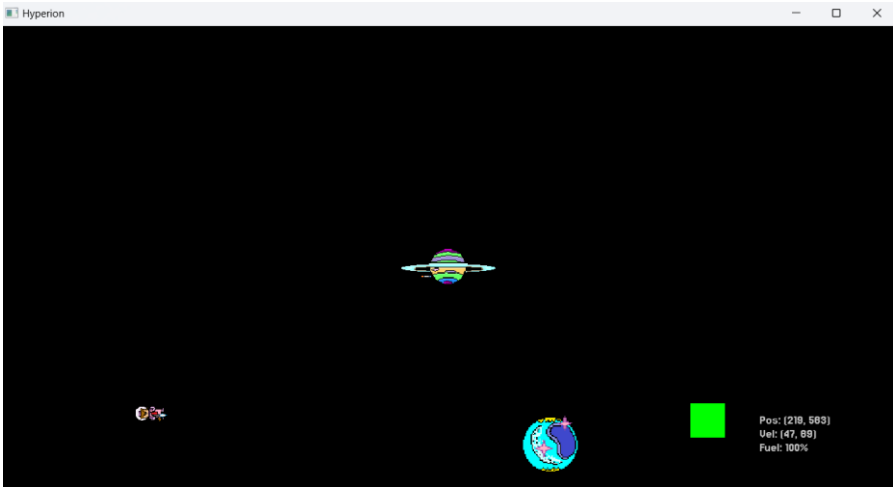
4. Test and Debugging

- **Main Menu**

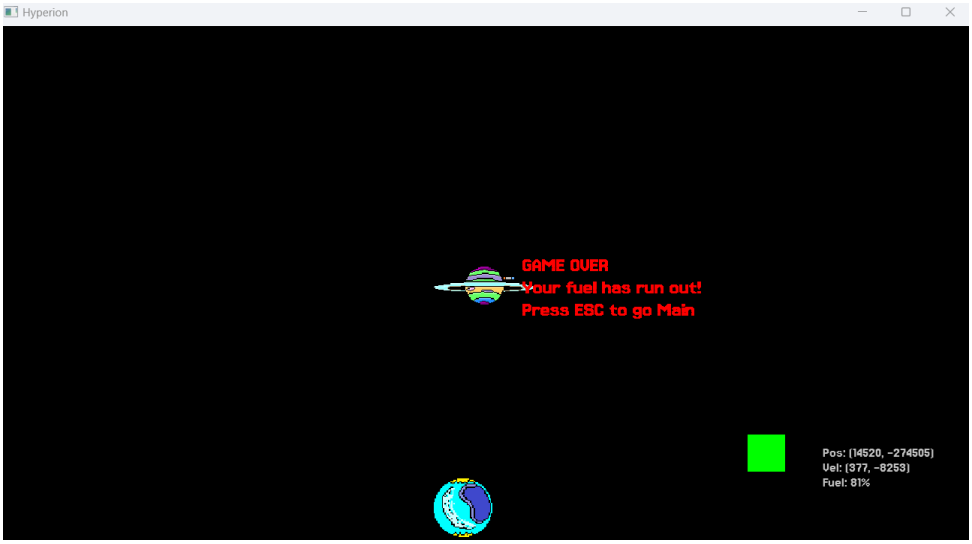
- **Info**



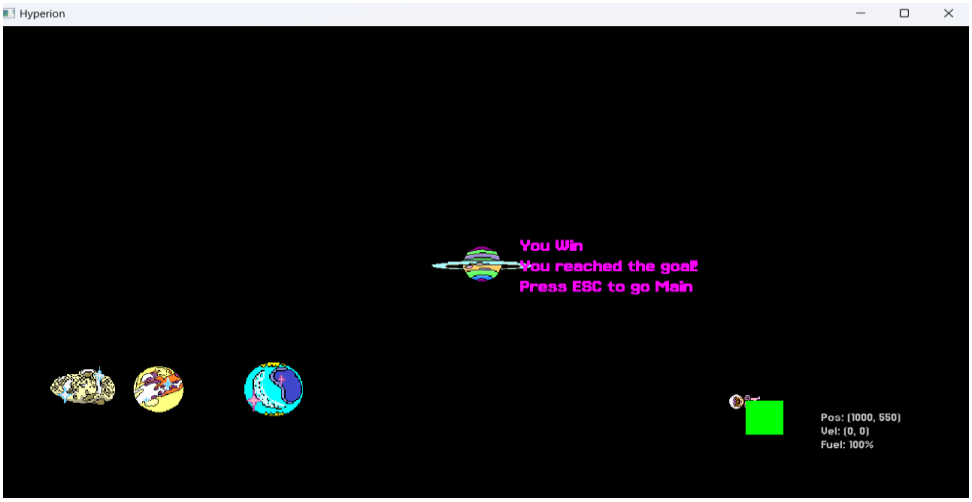
- **Game Play**



- **Game Play – Game over**



- **Game Play – Game win**



5. Conclusion

In conclusion, the development of the 2D graphic game "Escape the Loop" as part of a hackathon project provided a valuable learning experience in game design and programming. Utilizing C++ and the SFML library, we successfully created an engaging game that challenges players to navigate through a looping environment. The project allowed us to apply object-oriented programming principles, manage animations, and implement realistic physics, resulting in a cohesive and dynamic gaming experience. Through collaboration and iterative development, we overcame various challenges, enhancing our problem-solving skills and technical expertise. This project not only met the requirements of the hackathon but also received approval for submission as a final project for our programming course. Moving forward, we plan to explore additional features and improvements to further enhance the gameplay and user experience.

6. References

- **SFML Documentation:** https://www.sfml-dev.org/documentation/
- **C++ Standard Library Documentation:** https://en.cppreference.com/w/
- **Hackathon Guidelines and Resources**
- **Team Hyperion's internal Documentation and Meeting Notes**