Chapter Five: Implementation and Coding

**5.1 Implementation**

The objectives of building The Other Side has already been discussed and planned as a problem in the above chapters. But the implementation of the plan and the idea of how the final game looks like can be obtained from the below pictures.



Figure 5 Main menu

5 a shows the main menu of the game where the player is asked to select an option either to play the game or quit.

Let ‘Play’ be selected.

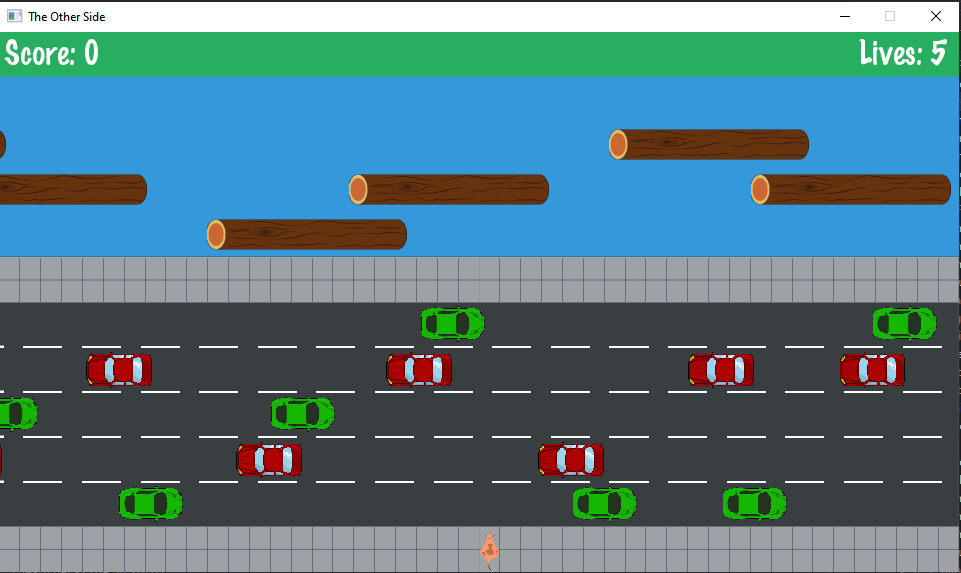


Figure 5 Game screen

5 b shows the game screen. Here we can see the kangaroo (player) at the bottom of the screen. There are 5 lanes on the road and the cars are moving in the left and right direction alternatively. There is a river which has 4 lanes and logs are floating on it. There is a partition between the road and the river. The player has to reach the top most row to win a score.

We can also see the score and lives on the top corners. As we can see, the player has 5 lives in total. After 5 lives, the game will be over.



Figure 5 Game over

After the player is hit by a car or drowns in the river 5 times, the game will be over. As we can see in the figure 5 c, the player can press space key to restart the game or esc to go back to the main menu.

**5.2 Coding of the game:**

In this game, C++ programming language is used. Since C++ is an Object Oriented Programming (OOP) language, we met our objective by writing different classes and interface. The classes of the project are as follows:

* AssetManager
* Collision
* Game
* GameOverState
* GameState
* InputManager
* Kangaroo
* Logs
* State
* MainMenuState
* SplashState
* StateMachine
* Vehicle

**AssetManager:**

AssetManager class stores all the textures and fonts in a map. Other classes can get their required texture and font using this class’s member functions.

Functions:

* *void LoadTexture ( std::string name, std::string filename )* : This function loads the texture from file and stores in a map object.
* *sf::Texture &GetTexture ( std::string name ) :* This function returns the texture using the name which is the key in the map.
* *void LoadFont ( std::string name, std::string filename ):* This function loads the font from file and stores in a map object.
* *sf::Font &GetFont ( std::string name ) :* This function returns the font using the name which is the key in the map.

Variables:

* std::map<std::string, sf::Texture> \_textures : This stores the textures in a key-value pair.
* std::map<std::string, sf::Font> \_fonts: This stores the fonts in a key-value pair.

**Collision:**

Collision class is responsible for detecting the collision between the player and vehicle or log.

Functions:

* bool CheckSpriteCollision ( sf::Sprite sprite1, sf::Sprite sprite2) : This function returns true if the two sprites passed to the function are colliding with each other.

**Game:**

Game class creates a window and controls the main loop of the game.

Functions:

* *Game(int width, int height, std::string title)*: This is the constructor of the Game class. It creates a window of the height and width with title passed to it.
* *void Run():* This function contains the main loop of the program. The loop stops only when the window is closed.

Variables:

* *GameDataRef \_data = std::make\_shared<GameData>( ):*  This variable stores the objects of StateMachine, Window, AssetManager, InputManager.
* *sf::Clock \_clock:* This is used to get the elapsed time of the game.
* *const float deltaTime = 1.0f / GAME\_FRAME\_RATE :* It stores the time the game takes to render a frame.

**GameOverState:**

GameOverState class is a derived class of State. This class handles when the player has lost all 5 lives. It receives the best score from the file and compares with the score of the player.

Functions:

* *GameOverState(GameDataRef data, int score) :*  This is the constructor of the class. The score earned by the player is passed to it.
* *void Init():* This function opens the best score file and compares the best score with the current score. If the current score is more than the best, the data in the file is overwritten. Also, the texts on the screen are initialized with fonts, position etc.
* *void HandleInput():* It handles the input in the gameover screen. If space is pressed, the game restarts. If ESC is pressed, the player is taken to the main menu.
* *void Draw(float deltaTime);* This function draws the texts, background onto the screen.

Variables:

* GameDataRef \_data; : This variable stores the objects of StateMachine, Window, AssetManager, InputManager.
* sf::Sprite \_background; : This stores the sprite for the background.
* sf::Sprite \_gameOverTitle; : This stores the sprite for the Game Over title.
* sf::Sprite \_gameOverBody; : This stores the sprite for the Game Over body.
* sf::Text \_playAgain; : This stores the sprite for the play again text.
* sf::Text \_backToMainMenu; : This stores the sprite for the back to main text.
* sf::Text \_scoreText; : This stores the sprite for the score text.
* sf::Text \_bestText; : This stores the sprite for the bests core text.
* int \_score; : This stores the score.
* int \_best; : This stores the best score from the file.

**GameState:**

This is the main class that controls everything in the playable part of the game. This class is also a derived class of the State interface.

Functions:

* *GameState(GameDataRef); :* This is the constructor of the GameState class.
* *void Init(); :* This functions initializes the state. It loads the audio, textures, score text, sets lives of the player and creates vehicles, logs and kangaroo object.
* *void HandleInput();* : This functions closes the window when the cross button is pressed.
* *void Update(float deltaTime); :* This function controls the spawning of the logs and vehicles.
* *void Draw(float deltaTime); :* This functions draws all the items to the screen.
* *void DetectCollision(float deltaTime); :* This functions checks collision between the kangaroo and vehicles/logs.
* *void addScore(); :*  This functions adds the score when the player reaches the end.

**InputManager:**

The InputManager class handles the input from the player. It handles the input using the following functions:

Functions:

* *bool IsTextHovered ( sf::Text object, sf::RenderWindow &window); :* It returns true when a text is hovered by the mouse cursor.
* *bool IsTextClicked(sf::Text object, sf::Mouse::Button button, sf::RenderWindow &window); :* It returns true when a text is clicked.
* *bool IsHovered ( sf::Sprite object, sf::RenderWindow &window); :* It returns true when a sprite is hovered by the mouse cursor.
* *bool IsSpriteClicked(sf::Sprite object, sf::Mouse::Button button, sf::RenderWindow &window); :* It returns true when a sprite is clicked.
* *sf::Vector2i GetMousePosition( sf::RenderWindow &window); :*  It returns the position of mouse cursor in the window.

**Kangaroo:**

Kangaroo class controls the player in the game. This class gets the input from the player and moves the player accordingly. It also manages the audio when the kangaroo moves.

Functions:

* *void update(float deltaTime); :* This function checks the input from the player and moves the kangaroo in the game.
* *float getPosition(); :* This function returns the current position of the kangaroo in the game.
* *void setPosition(float yPosition); :* It sets the position of the kangaroo in the game.
* *void drawKangaroo(); :* It draws the sprite of kangaroo in the screen.
* *void Attach(float Xvelocity); :*  It attaches the kangaroo to the log and moves the kangaroo with it.

**Log:**

Log class controls the logs in the river. This class spawns the logs randomly in the river and moves them in a certain direction according to their lane position.

Functions:

* *void spawnLogsLeft(float Lane); :*  This functions spawns the logs in the left side.
* *void spawnLogsRight(float Lane); :*  This functions spawns the logs in the right side.
* *void moveLogsLeft(float deltaTime); :* This function moves the logs in the left direction.
* *void moveLogsRight(float deltaTime); :* This function moves the logs in the right direction.
* *int getXvelocity(); :*  It returns the velocity of the logs.
* *void drawLogs(); :*  It draws the logs in the screen.

**State:**

State is an abstract class. This is an interface from which other state classes are derived. The virtual functions are as follows:

* virtual void Init( ) = 0;
* virtual void HandleInput( ) = 0;
* virtual void Update( float deltaTime ) = 0;
* virtual void Draw( float deltaTime ) = 0;

**MainMenuState:**

MainMenuState is the state that comes after SplashState in the program. This is a display state where user can choose between two options, play and quit. Play option takes user to the main GameState, whereas the quit button closes the program.

Functions:

* *void Init():* This is to load textures, sound and fonts to set properties
* of attributes of this class.
* *void HandleInput()* : This handles any events occuring on the screen such as minimize, resize and close. In this case, it also handles clicks for the buttons displayed on the main menu. If play is clicked, it creates a new state called GameState.
* *void Update():* This method updates buttons of they are hovered. It also changes the buttons color if mouse is hovered above these buttons.
* *void Draw():* This method draws all the sprites, and buttons on the window.

**SplashState:**

SplashState is a state that inherits from the abstract class State. It only appears on the screen for provived duration. It displays an image for certain time and it ends.

Functions:

* *void Init():* This is to load image for the splashState and set its texture.
* *void HandleInput() :* This handles any events occuring on the screen such as minimize, resize and close.
* *void Update():* This method processes all the changes that occurs on the screen. In splashState, there is no changes so nothing to update here.
* *void Draw ():* This method draws the image to the window.

**StateMachine:**

StateMachine is a class that creates a state of game such as splashState, MainMenuState, GameState and GameOverState. Basically, it is a class that create new pages and only one page can be displayed at a time.

Functions:

* *void AddState ( StateRef newState, bool isReplacing = true):*This method is used to add a new state to the program. It takes name of new state as a parameter, and it replaces the previous state, so isReplacing is set to true. New state is added to top of the stack.
* *void RemoveState ( ) :* This method is used to set isRemoving boolean as true when called.
* *void ProcessStateChanges ():* This method monitors all the activities of states. Actual adding and removing of states takes place here. StateRef GetActiveState(): This method returns the active state i.e. state present on the top of stack to perform operations on it.

**Vehicle:**

Vehicle class is the class that is responsible for all the car sprites drawn onto the screen. There are total 5 lanes on the background image and vehicle spawning is done out of which red car spawns on 3 lanes, namely 1st , 3rd and 4th lane from bottomgreen car spawns on the remaining two. On the basis of direction of spawning of vehicle, there are two object pointers created in the gameState, one to spawn vehicle from left and other from right.

Functions:

* *Vehicle Constructor*: This constructor sets its \_data attribute with the shared data of type GameDataRef
* *void spawnVehicleLeft ( float lane ):* spawnVehicleLeft spawn vehicle starting from the left of the screen . It takes one argument 'lane' which is a random number generated in the gameState and selects lane accordingly.
* *void spawnVehicleRight ( float lane):* spawnVehicleRight spawns vehicle starting from the right side of the screen. It also selects lane based on the random number passed to it from gameState.
* *void moveVehicleLeft ( float deltaTime ):* This method moves vehicle to the left and gives visualization of moving vehicle.
* *void moveVehicleLeft ( float deltaTime ):* It is pretty much the same as moveVehicleLeft. The only difference is it moves vehicle to right direction.
* *const std::vector< sf::Sprite > &getVehicleSprite( ) const:* This method is necessary for collision detection. When this method is called, it returns vehicle sprite which is used to check collision with the player object.