POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Games and Multimedia

Advanced Game Programing Topics

Report AGPT

Professor: Gustavo Reis

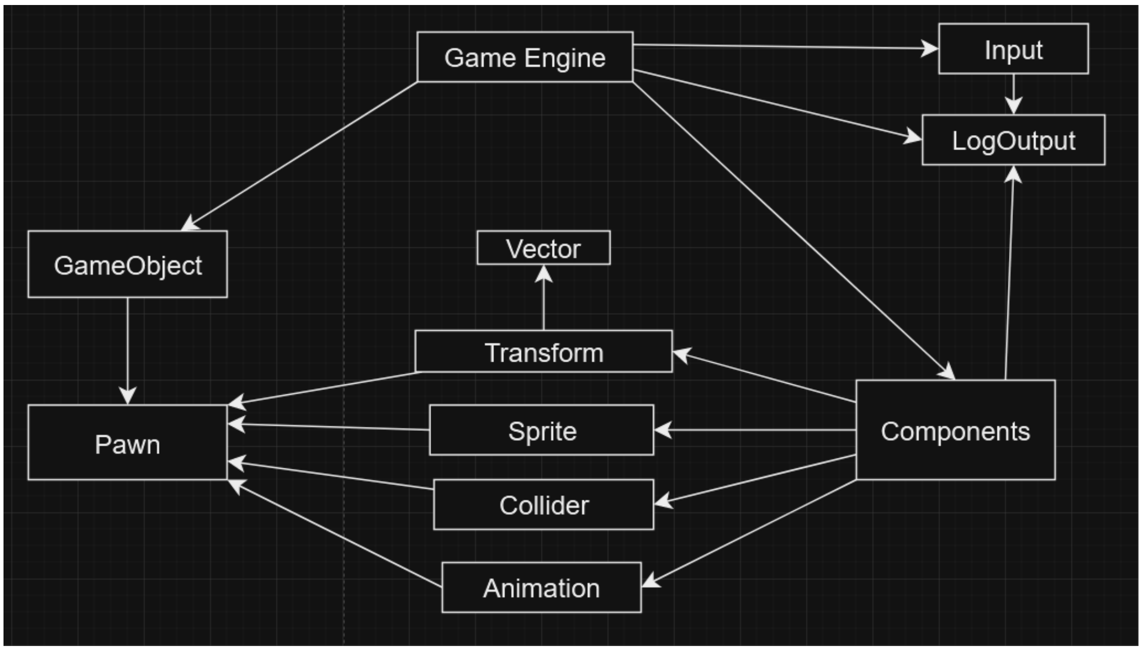
Alexandre Rodrigues

2200728

Index

Engine Logic 3
Handle Events 3
Components 4
  Transform Component 4
  Sprite Component 4
  Collider Component 5
  Animation 5
Game Object 5
Pawn 5
Game Logic 6
Game Manager 6
Player 7
Pickups 8
  Power up 8
  Shield 8
Enemies 8
  Rusher 8
  Loner 9
Asteroids 9
Rocks 9
UI 9
Objectives 10

Engine Logic



The Engine starts with the GameEngine.h / GameEngine.cpp files.It acts as the central piece for initializing, running and shutting down the game. In order to begin the world in which we set gravity and contact listeners, it first initializes the SDL Classes (Wrapper, Window, and Renderer). Next, on the engine loop, the following processes are executed:

- DeltaTime calculations
- Handle Events (Inputs)
- Update Objects on world
- Render
- Update Surface

Handle Events

1. Input Handling:

- The Input class supports both keyboard and gamepad input.
- It uses the SDL_GetKeyboardState function to track the keyboard state**.**
- For gamepads, it initializes the relevant SDL subsystem and checks for connected controllers.
- If a gamepad is found, it opens it and stores a pointer for accessing button and axis data.

2. Event-Driven Approach:

- The Listen function serves as the main input handling loop.

- It uses an SDL_Event struct to poll for and analyze incoming events.
- Based on the event type, it calls specific functions to handle key presses/releases, gamepad button presses/releases, and joystick axis movements.

3. Input Checking Functions:

The class provides various functions to query the current input state:

- GetKeyDown and GetKeyUp check for key presses and releases.
- GetButtonDown and GetButtonUp check for gamepad button presses and releases.
- GetAxis reads the value of a specific gamepad axis.

Components

Transform Component

The TransformComponent class includes vector2d header and provides a foundation for managing an object's position and movement.

Sprite Component

- **Texture Management:**
  - Loads textures from file paths.
  - Allows setting and swapping textures dynamically.
- **Transform Integration:**
  - Retrieves position and size information from the linked TransformComponent.
  - Updates its drawing position based on transform data.
- **Drawing and Blending:**
  - Renders the texture to the screen using Texture class functionality.
- **Animation Support:**
  - Creates and stores named animations with frame ranges and speeds.
  - Plays specific animations by name, updating frame index and speed.
- **Flashing Effect:**
  - Enables a temporary flashing effect with optional color.
  - Controls flashing state and duration.

Collider Component

- Creates a dynamic Box2D body based on the provided width and height parameters.
- Attaches a Box2D shape representing the collider to the body.
- Sets up a fixture with specific properties like density, friction, and sensor status.
- Positions the body based on the parent entity's transform data.

- Continuously synchronizes the collider's position and rotation with the parent entity's transform, ensuring accurate collision detection.
- Checks for entity destruction and removes the collider from the physics world if necessary.

Animation

The animation structure serves as a simple way to manage animation data by providing the index, frames, and speed of which we want the animation to play.

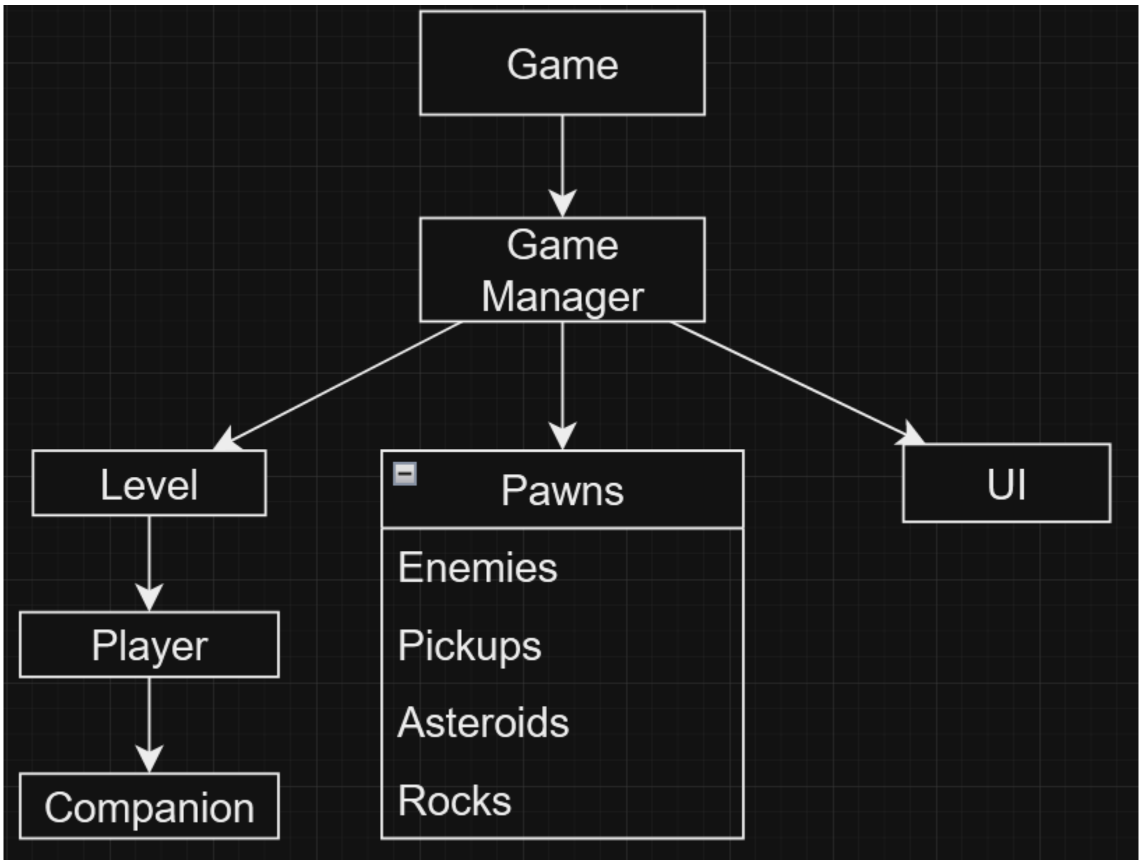
Game Object

Serves as a fundamental building block for pawns within the engine and provides basic lifecycle management like initialization, update and destroyed.

Pawn

Represents an entity in the game environment like a player or enemy and provides max health, current health, and a custom function to destroy the object and spawn an explosion on its position.

Game Logic



The game starts with game.cpp, which includes the GameEngine.h header file. This allows access to all engine functionalities, then creates an instance of the GameEngine class, to initialize the game window and setting up the game world. After the initialization we create an instance of Game Manager, which takes on the responsibility of managing all the different aspects of the game experience.

Game Manager

This is the most important class of our game since it manages the gameplay of the game.

- **Create Level**
  - Initializes the game world by creating a Level entity using the world object.
- **Create Player**
  - Instantiates the player and companion entities.

- **Spawn Enemies**
    - Tracks elapsed time using timers for Loner and Rusher enemies.
    - Randomly determines the number of enemies to spawn within preset ranges.
    - Creates the specified number of enemies using the world.CreateEntity method.
- **Spawn Pickups**
    - Uses a timer to control the spawn rate of WeaponPickup and ShieldPickup entities.
    - Creates multiple pickups at each spawn event.
- **Spawn Asteroids**
    - Like enemy spawning, controls the frequency and quantity of BigAsteroid and MetalAsteroid entities.
- **Spawn Rocks**
    - Utilizes a random timer for periodic spawning of the Rocks entity.
- **Load UI**

Player

- **Initialization:**
    - Sets its initial position and speed.
    - Adds necessary components for visuals, movement, and collision detection.
    - Configures animations for idle and turning states.
    - Spawns and stores companion entities.
- **Update:**
    - Handles movement based on player input and updates position accordingly.
    - Animates the player sprite based on movement direction.
    - Controls firing mechanics, including weapon type and cooldown.
    - Checks for collisions with other entities and triggers appropriate actions (damage, flashing visual).

- **Firing:**
    - Determines if firing is allowed based on cooldown.
    - Creates projectiles of the appropriate type based on current weapon augmentation.
- **Interactions:**
    - Takes damage upon collision with enemies, asteroids, or enemy projectiles.
    - Updates health and visuals accordingly.
    - Manages player lives.
    - Can receive health boosts up to the maximum health limit.

Pickups

Pickups are randomly spawned and move constantly towards the player.
There are two types of pickups, power up and Shield.

Power up

When the player or companions enter in contact with these their weapons are upgraded sequentially, offering three tiers, (default, medium, heavy) once the weapons reach the heavy upgrade they can't be further upgraded.

Shield

When this type of pickup enters in contact with the ship or the companions they get a small health boost.

Enemies

Rusher

- **Initialization:**
    - Randomly determines its initial position.
    - Adds necessary components for movement, visuals, and collision detection.
    - Configures an idle animation for the enemy sprite.
    - Sets tags for identification and interaction.
    - Randomly chooses a movement speed within a set range.

- **Update:**
  - Uses its start position to determine initial movement direction (left or right).
  - Continuously moves horizontally until reaching the opposite edge of the screen.
  - Checks for boundary collision and destroys itself upon reaching the edge.
- **Interactions:**
  - Detects collisions with projectiles and triggers a flashing visual effect.
  - Takes damage upon collision with projectiles and tracks its health.
  - Destroys itself upon reaching zero health.

Loner

Same spawn logic as the rushers but instead of horizontal movement they move vertically and can fire projectiles at the player.

Uses a timer to control the firing rate of projectiles and creates an EnemyProjectile entity moving horizontally towards the player.

Asteroids

Big Asteroids are randomly spawned with random velocity at the end of the screen and move horizontally towards the player, if the player shoots and destroys the big asteroids they break into three medium sized asteroids moving in random directions, if they are destroyed, they break into three small asteroids continuously moving in random directions as well, if these are destroyed they simply create an explosion and dissipates.

There are metal asteroids as well that are unbreakable thus the player must simply avoid those.

If any asteroid enters in contact with the player or the companions, they take damage.

Rocks

Background rocks are spawned randomly on top or bottom of the screen and move slowly horizontally, they don’t damage either the player or the companions.

UI

At the start of the game the UI is created showing the number of lives of the player displayed with the sprite of the ship on the left corner of the window.

Objectives

Every major objective was achieved but there were some minor implementations that could be improved like player lives UI update when killed, spawn companions when picking up upgrades (if they are dead), the animations for the asteroids or having defined layers in order for the background rocks don’t be on top of the ship sprite.