Vampire Survivest Documentation

Created by

Thanapat Ussawanarong 6430158021

Teme Sutichairatana 6430174021

2110215 Programming Methodology

Semester 1 Year 2022

Chulalongkorn University

Vampire Survivest

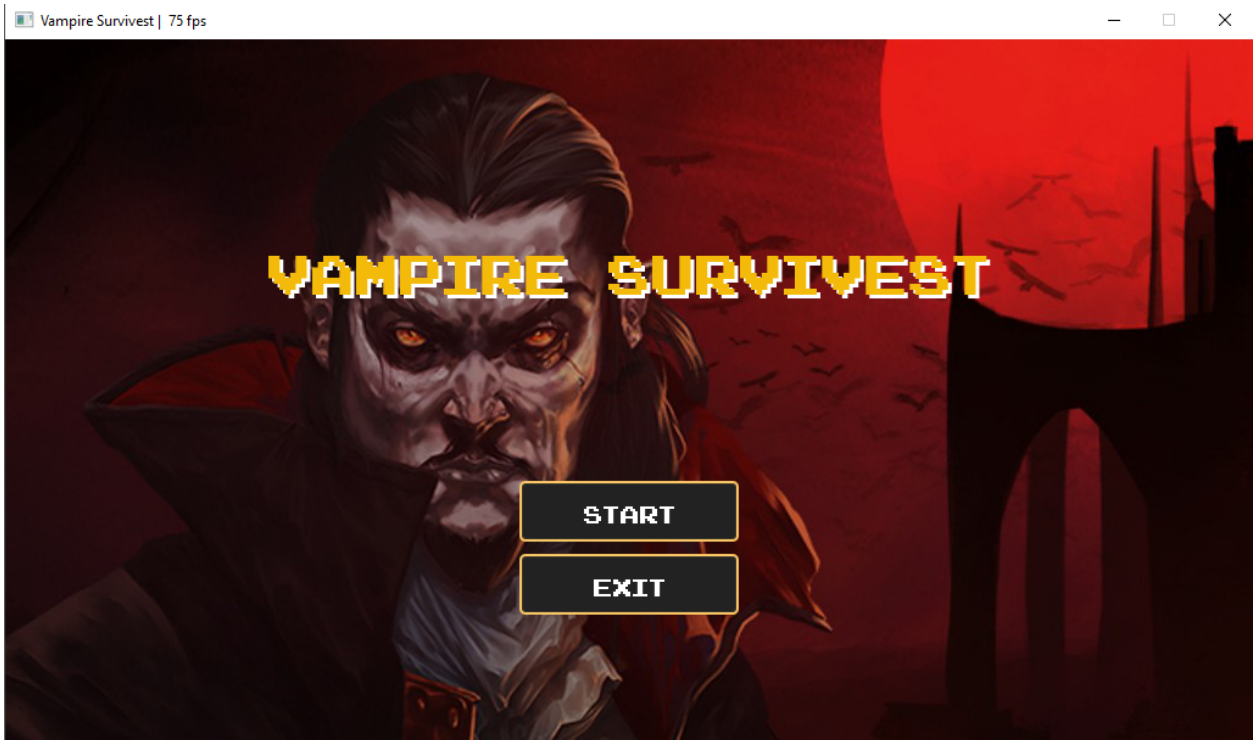
Introduction

Vampire Survivest is an action roguelite game that has been created with an inspiration from a famous video game “Vampire Survivor”.

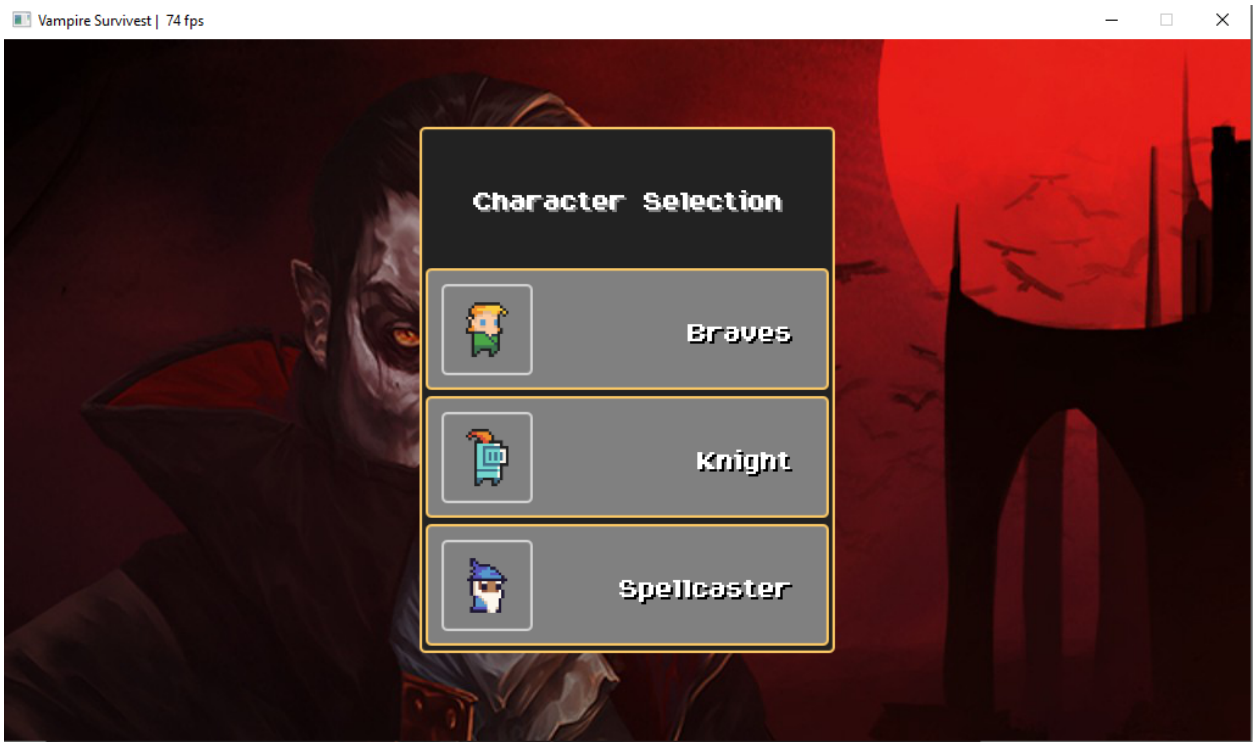
Gameplay

In Vampire Survivest, the player selects one of the game’s playable characters and enters an abandoned world where the player’s goal is to survive against hordes of approaching enemies. At the beginning of the game, the player is granted a base weapon of their character to fight back at the enemies. By killing enemies, the player gains experience by collecting coins dropped from enemies’ corpses and can level up, allowing them to upgrade their abilities and powers to help them survive the tougher foes yet to come.

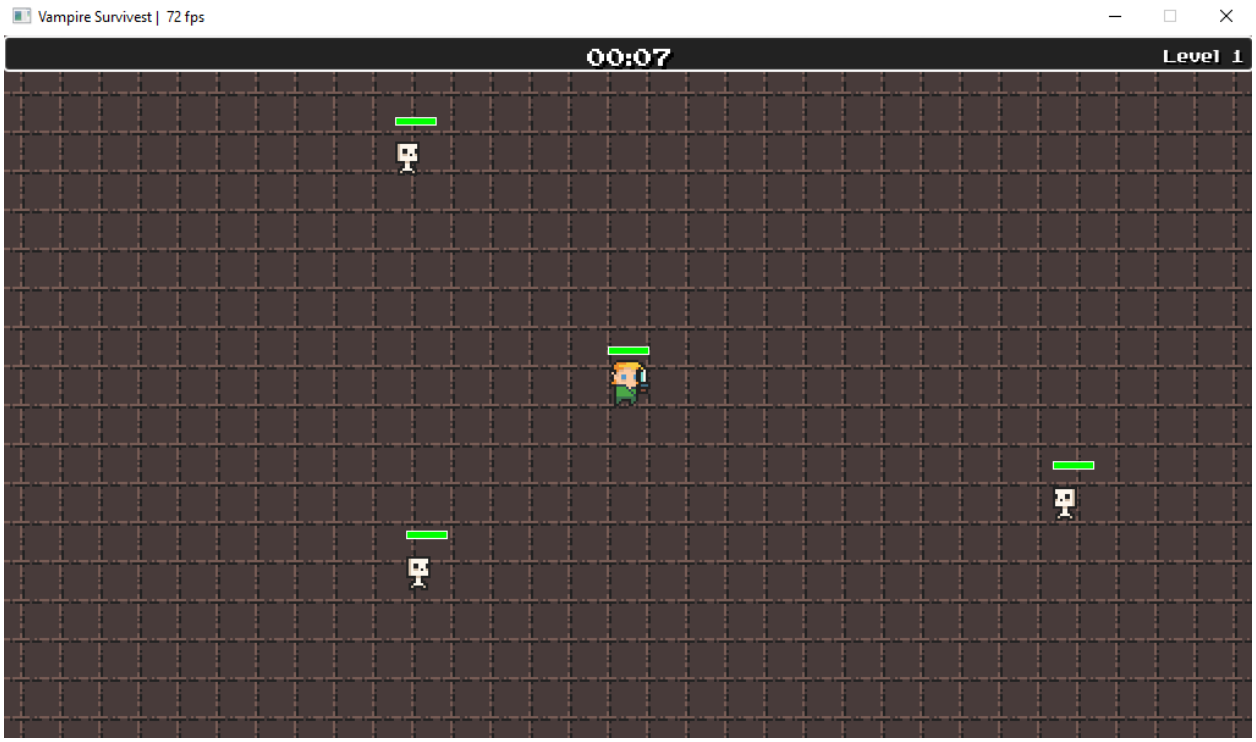
Examples of the game are on the next pages



Main Menu Scene

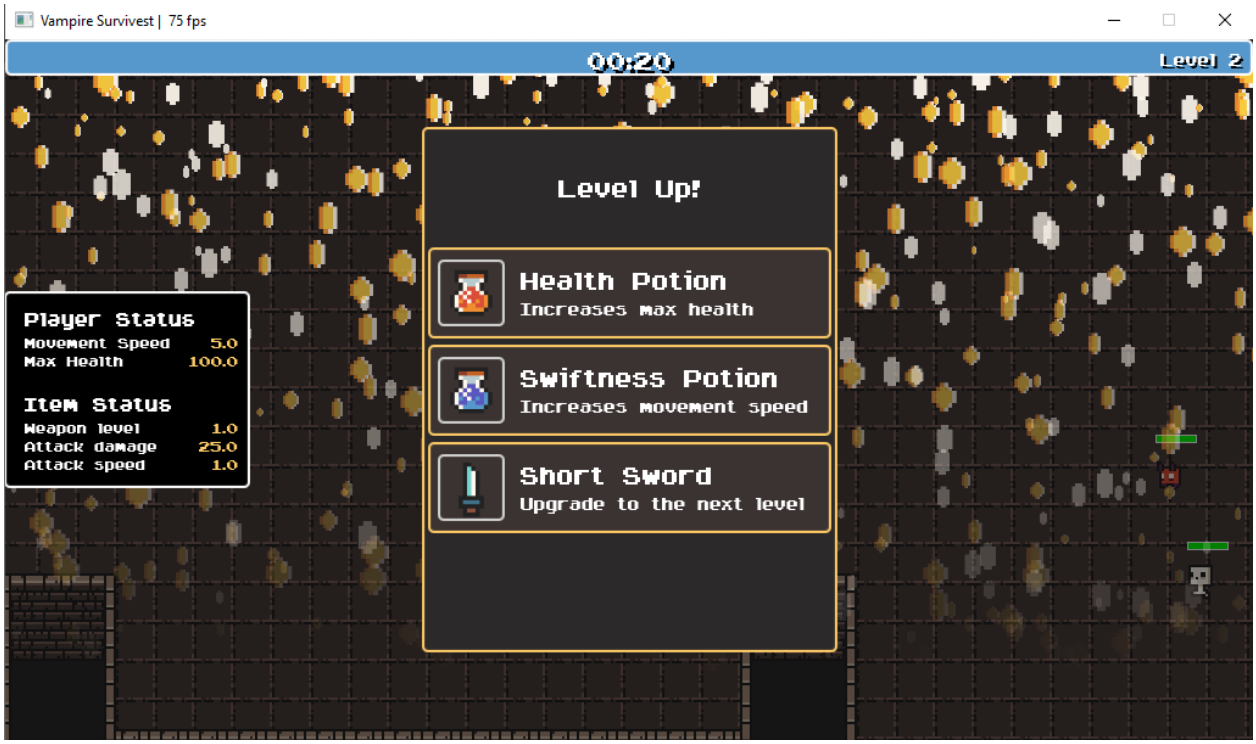


Character Selection

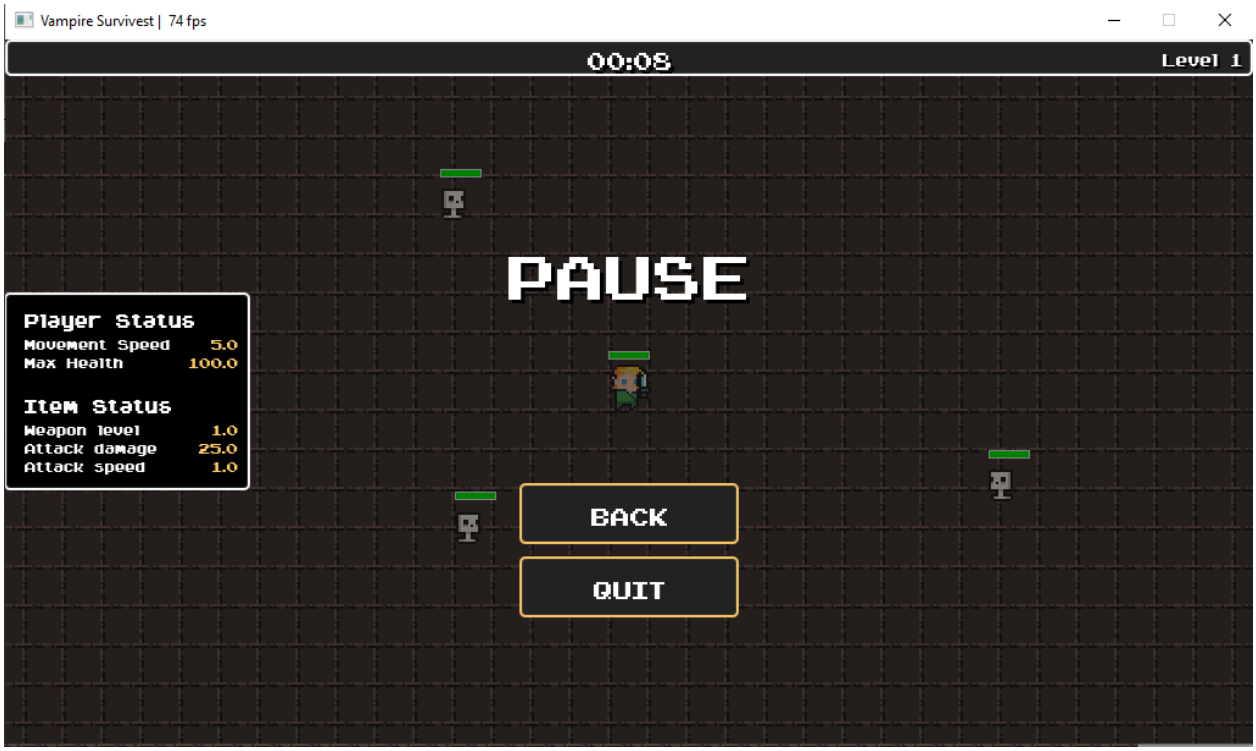


In-game Scene

In-game coin dropped



Upgrade screen

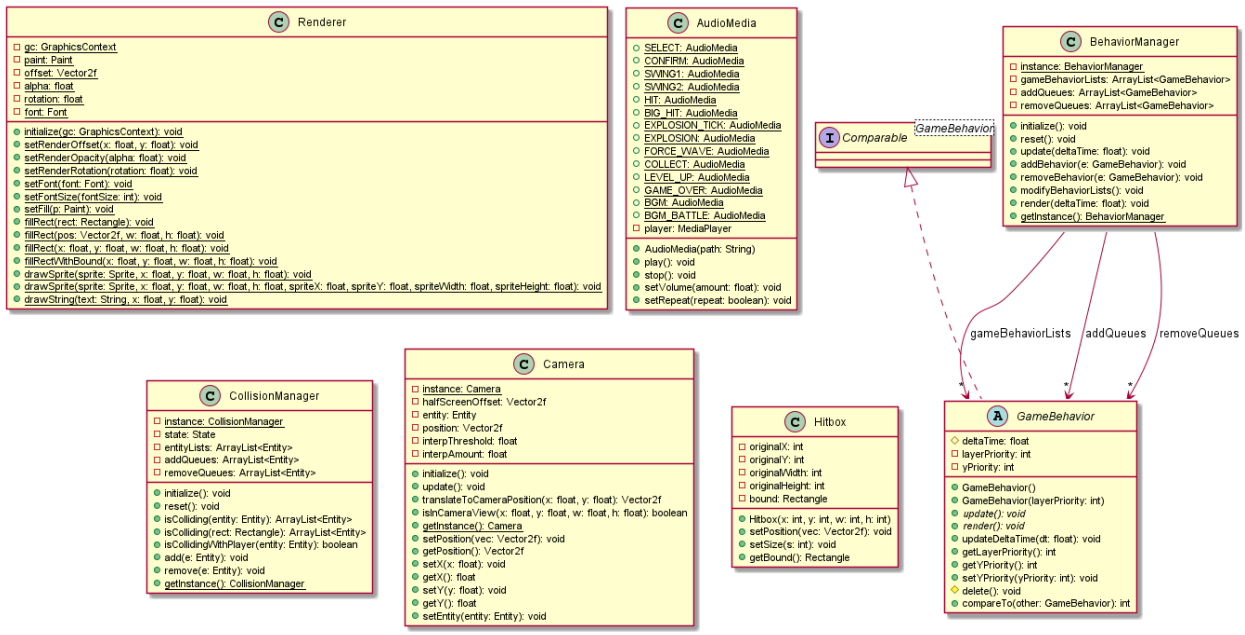


Pause screen

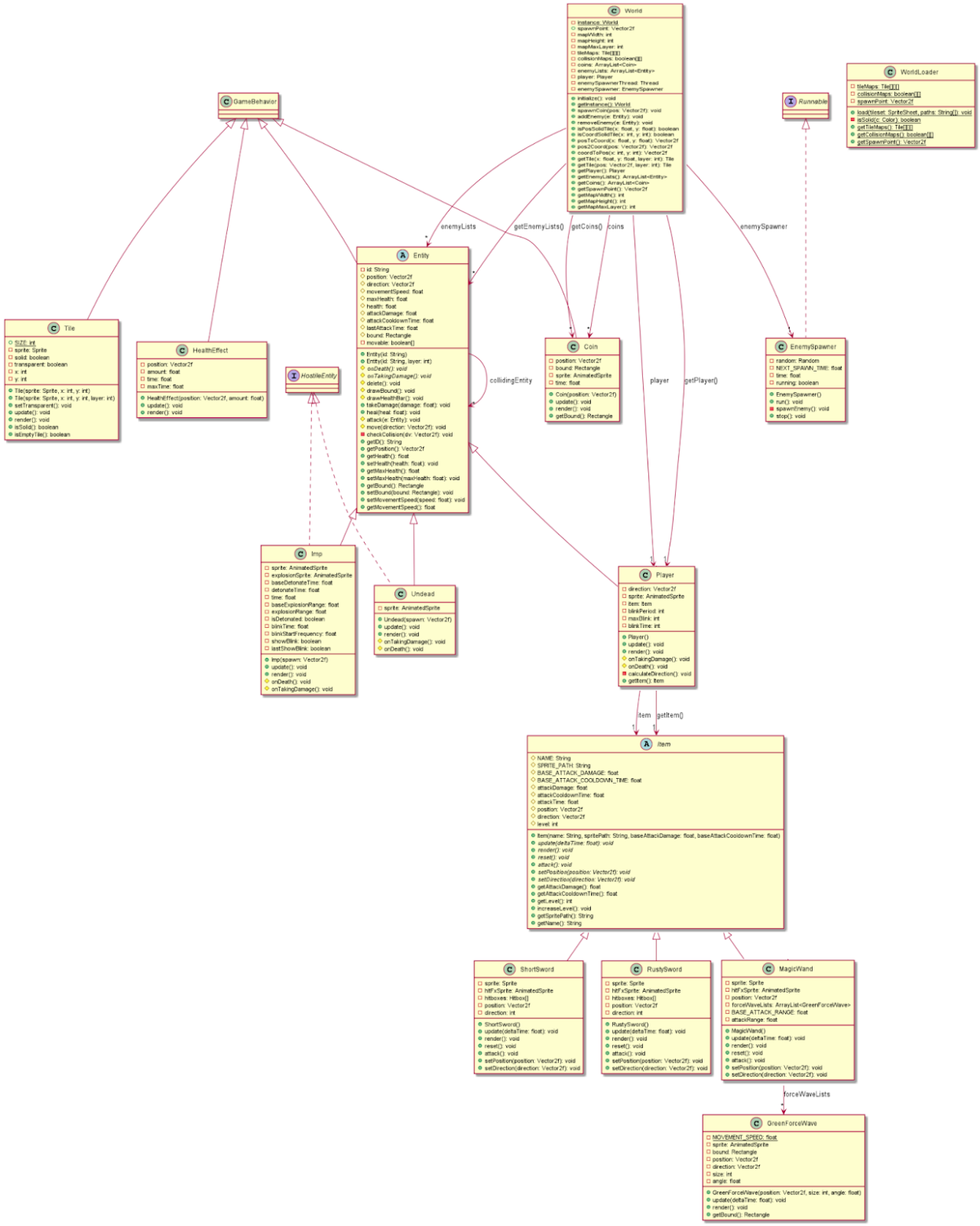
Game Over

Class Diagram

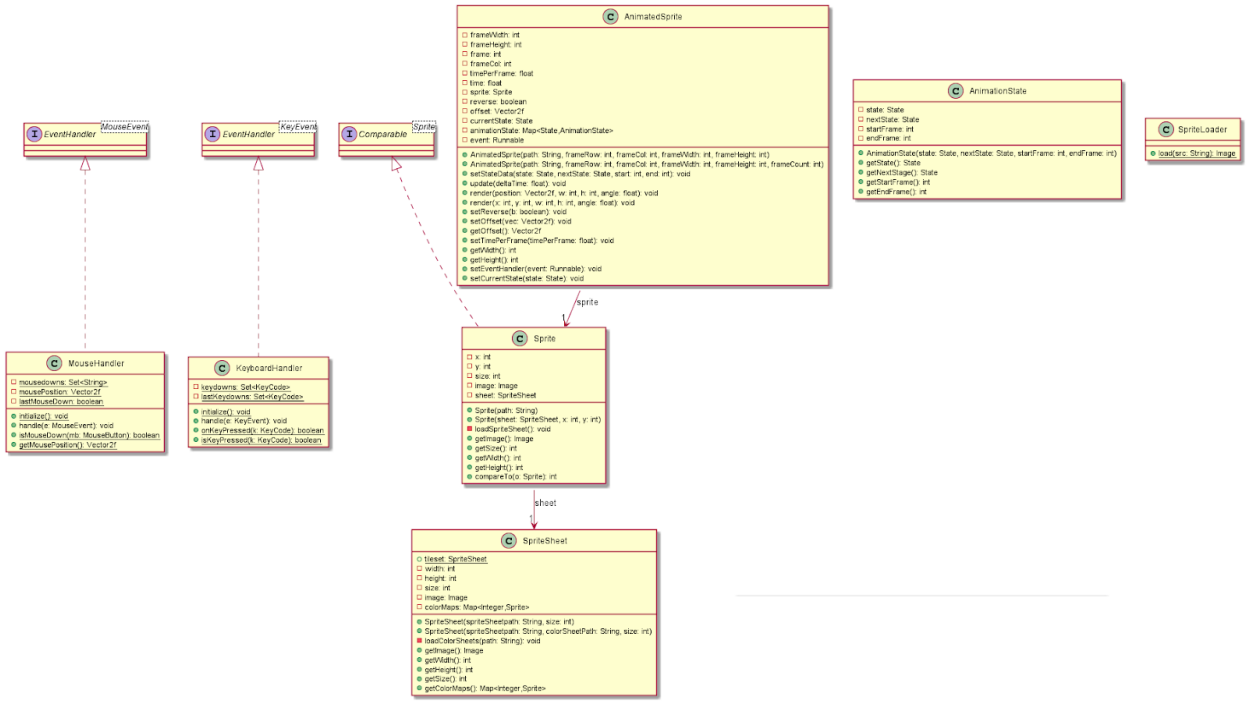
package core (video, audio, manager)



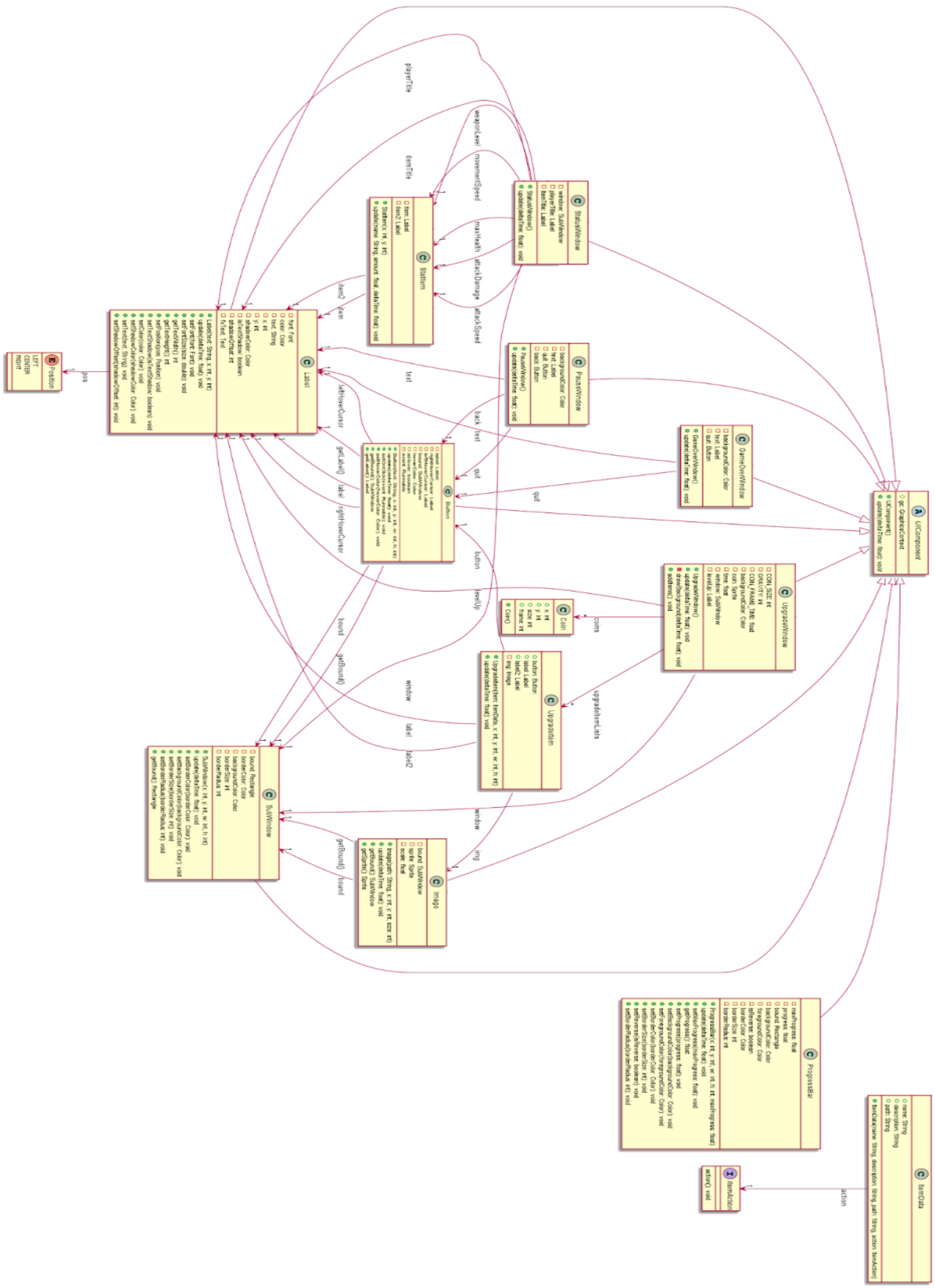
package core.game



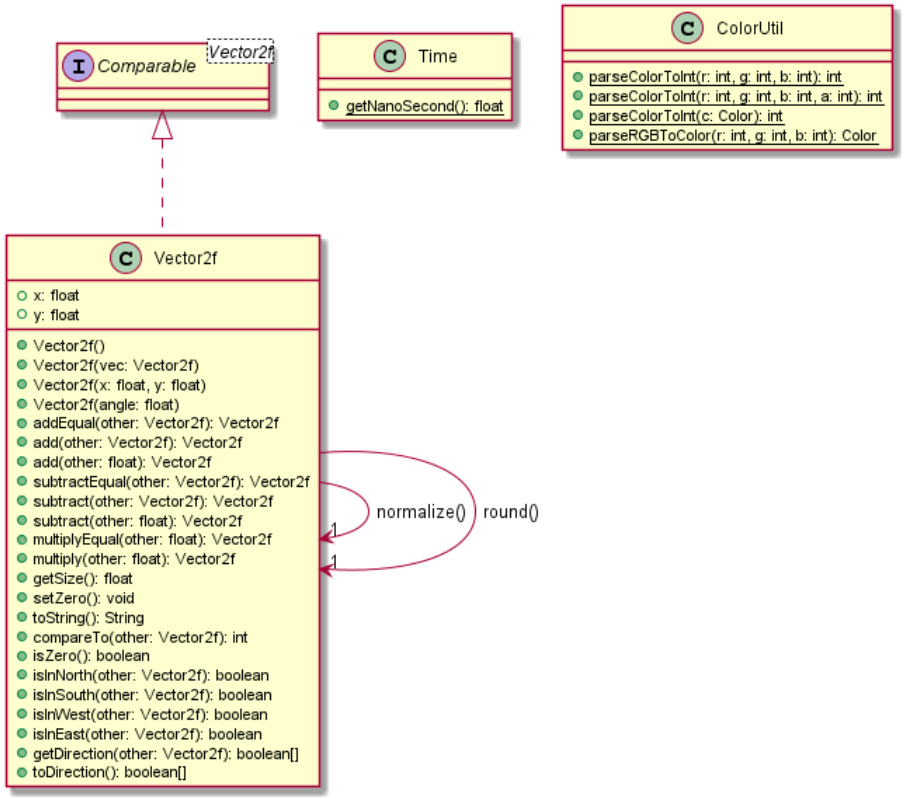
package core.inputHandler and core.sprite



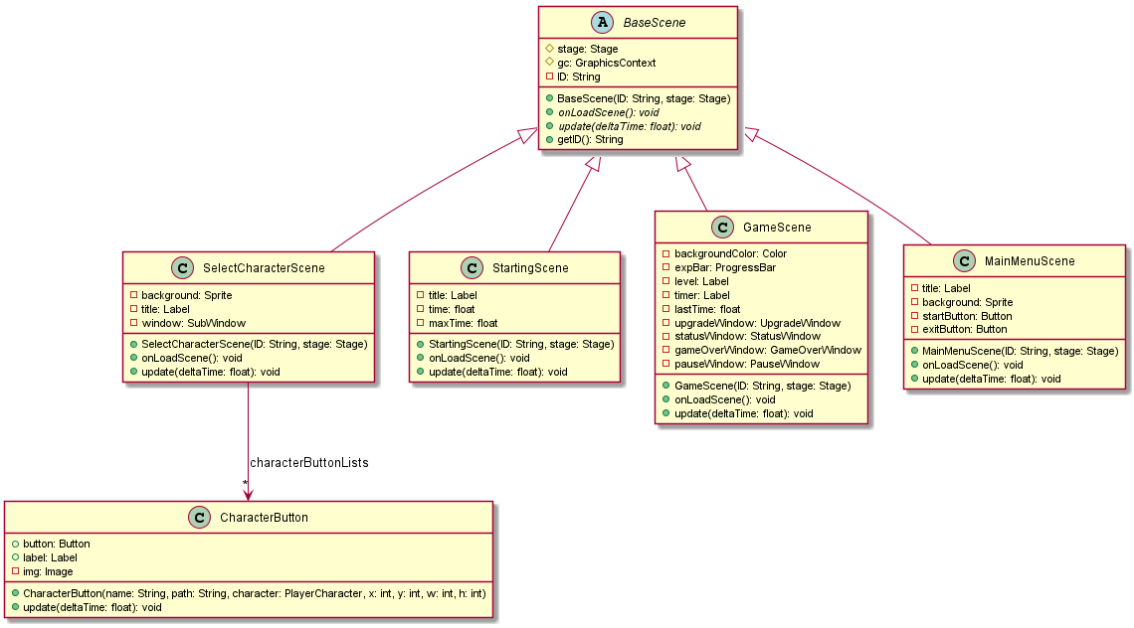
package core.ui



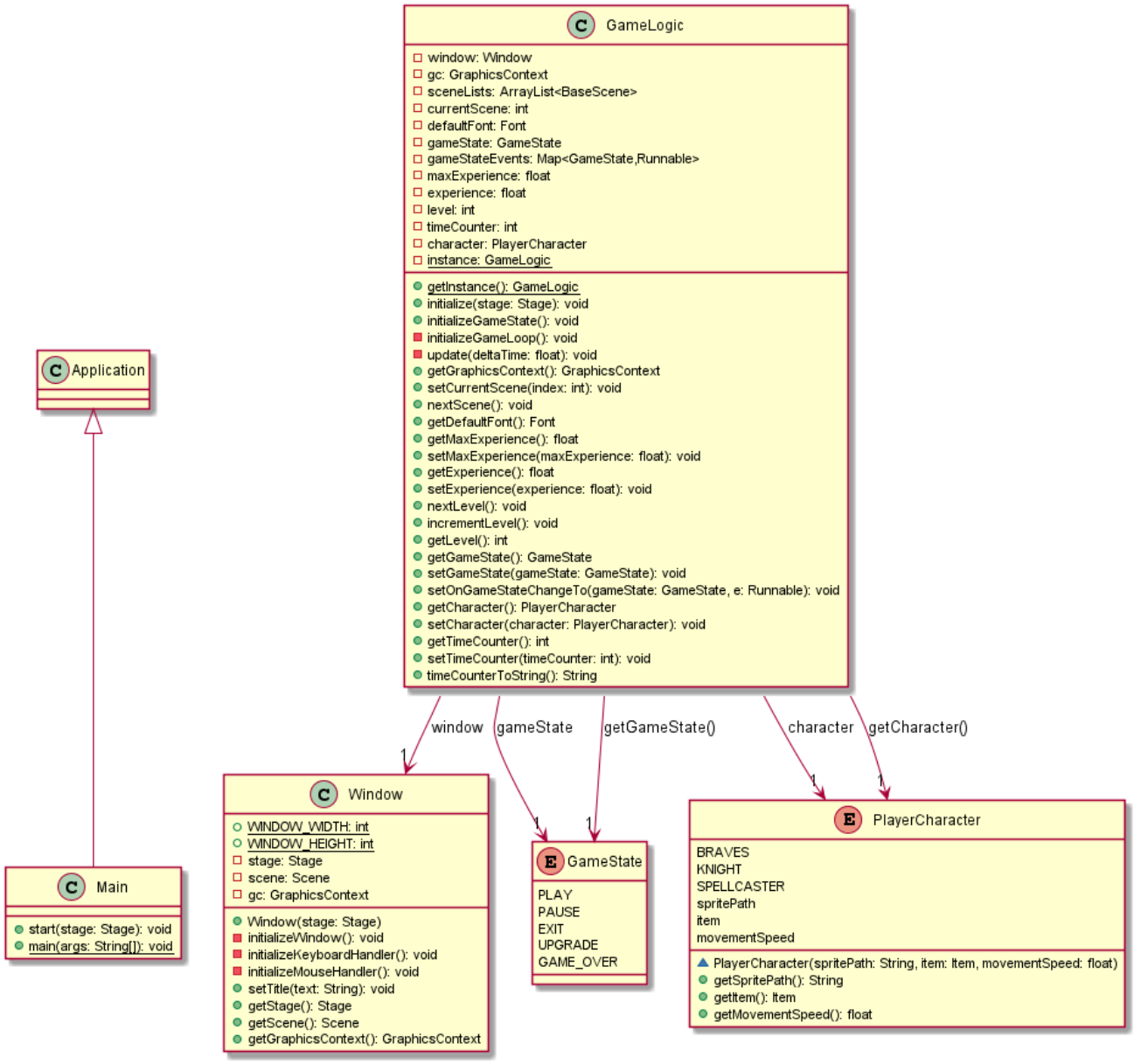
package util



package scene



package logic



1. package core

1.1. class Camera

This class represents the camera or frame that is shown on the user's screen.

1.1.1. Fields

- Camera instance	Singleton pattern
- Vector2f halfScreenOffset	Camera’s position relative to world
- Entity entity	The entity to be updated
- Vector2f position	The entity’s position
- float interpolationThreshold	Set the value to 0.05f
- float interpolationAmount	Set the value to 0.045f

1.1.2. Methods

+ void initialize()	Initialize halfScreenOffset and position
+ void update()	Update the entity’s position
+ Vector2f translateToCameraPosition(float x, float y)	Return a position relative to camera view
+ boolean isInCameraView(float x, float y, float w, float h)	Return true if the parameter’s position is in the camera view
+ Camera getInstance()	Singleton pattern
Getter/Setter for position and entity	

1.2. class Renderer

This class includes methods that will make changes to the GUI.

1.2.1. Fields

- GraphicsContext gc	The game's graphics context
- Paint paint	Paint value
- Vector2f offset	Offset value
- float alpha, rotation	Alpha and rotation value
- Font font	The game's font

1.2.2. Methods

+ void initialize(GraphicsContext gc)	Initialize all fields
+ void setFill(Paint p)	Set the fill color for GraphicsContext
+ void fillRect(float x, float y, float w, float h)	Fill the graphics context according to the parameters' values
+ void fillRectWithBound(float x, float y, float w, float h)	Fill the graphics context white and draw a boundary according to the parameters' values
+ void drawSprite(Sprite sprite, float x, float y, float w, float h, float spriteX, float spriteY, float spriteWidth, float spriteHeight)	Draw a sprite on the graphics context

+ void drawString(String text, float x, float y)	Draw a text on the graphics context
Setter for offset, alpha, rotation and font	

2. package core.audio

2.1. class AudioMedia

This class is used to load and store all of the audio sources.

2.1.1. Fields

+ AudioMedia SELECT	Stores an audio from "audio/select.wav"
+ AudioMedia CONFIRM	Stores an audio from "audio/confirm.wav"
+ AudioMedia SWING1	Stores an audio from "audio/swing_1.wav"
+ AudioMedia SWING2	Stores an audio from "audio/swing_2.wav"
+ AudioMedia HIT	Stores an audio from "audio/hit.wav"
+ AudioMedia BIG_HIT	Stores an audio from "audio/big_hit.wav"
+ AudioMedia EXPLOSION_TICK	Stores an audio from "audio/08_Step_rock_02.wav"
+ AudioMedia EXPLOSION	Stores an audio from "audio/25_Wind_01.wav"

+ AudioMedia FORCE_WAVE	Stores an audio from "audio/04_Fire_explosion_04_medium.wav"
+ AudioMedia COLLECT	Stores an audio from "audio/collect.wav"
+ AudioMedia LEVEL_UP	Stores an audio from "audio/level_up.wav"
+ AudioMedia GAME_OVER	Stores an audio from "audio/88_Teleport_02.wav"
+ AudioMedia BGM	Stores an audio from "audio/Goblins_Den_(Regular).wav"
+ AudioMedia BGM_BATTLE	Stores an audio from "audio/Goblins_Dance_(Battle).wav"
- MediaPlayer player	Provides the control for playing the media

2.1.2. Constructor

+ AudioMedia(String path)	Initialize a new media by calling ClassLoader and set the volume to 0.3

2.1.3. Methods

+ void play()	Play the media player
+ void stop()	Stop the media player
+ void setVolume(float amount)	Set the volume to match the amount

+ void setRepeat(boolean repeat)	Set the repetition of the audio

3. package core.behavior

3.1. abstract class GameBehavior implements Comparable<GameBehavior>

This class is a base class for all game objects which provides other classes update, render and delete methods.

3.1.1. Fields

# float deltaTime	Difference between latest update time and current time
- int layerPriority, yPriority	Priority value of each layer

3.1.2. Constructor

+ GameBehavior(int layerPriority)	Initialize all fields and add this behavior to the behavior list

3.1.3. Methods

+ *void update()*	An abstract method for updating the object
+ *void render()*	An abstract method for rendering the object
+ void updateDeltaTime(float dt)	Set deltaTime to match the parameter

# void delete()	Remove the behavior from the behavior list
+ int compareTo(GameBehavior other)	Compare this object's priority to other
Getter for all fields and Setter for YPriority	

3.2. class BehaviorManager

This class is used to manage all game behaviors, including updating and rendering the game behaviors as well.

3.2.1. Fields

- <u>BehaviorManager instance</u>	Singleton pattern
- ArrayList<GameBehavior> gameBehaviorLists, addQueues, removeQueues	A list of all game behaviors and queues to prevent iterator invalidation problems

3.2.2. Methods

+ void initialize()	Initialize all lists in the fields
+ void reset()	Clear all lists in the fields
+ void update(float deltaTime)	Update the camera and all game behaviors in the list. Also adjust the queue elements properly.
+ void addBehavior(GameBehavior e)	Add the behavior to addQueue

+ void removeBehavior(GameBehavior e)	Add the behavior to removeQueue
+ void modifyBehaviorLists()	Add all the behavior awaited in the addQueue to the gameBehaviorLists. Then remove all behavior in the removeQueue from the gameBehaviorLists.
+ void render(float deltaTime)	Render the game behavior
+ BehaviorManager getInstance()	Singleton pattern

4. package core.collision

4.1. class Hitbox

This class represents the rectangle-shaped hitbox of the entity.

4.1.1. Fields

- int originalX, originalY, originalWidth, originalHeight	A boundary of a hitbox
- Rectangle bound	A rectangle bound that represents a hitbox

4.1.2. Constructor

+ Hitbox(int x, int y, int w, int h)	Initialize all fields

4.1.3. Methods

+ void setPosition(Vector2f vec)	Set bound's position
+ void setSize(int s)	Set bound's width and height
+ Rectangle getBound()	Return the hitbox

4.2. class CollisionManager

This class is used to manage all the entity's collisions in the game.

4.2.1. Fields

- <u>CollisionManager instance</u>	Singleton pattern
enum State	Holds the following states: IDLE, RUNNING
- State state	Set the state to IDLE
- ArrayList<Entity> entityLists, addQueues, removeQueues	A list of all entities in the game and queues to prevent iterator invalidation problems

4.2.2. Methods

+ void initialize()	Initialize all lists in the fields
+ void reset()	Clear all lists in the fields
+ ArrayList<Entity> isColliding(Entity entity)	Call another method to check the entity's colliding state
+ ArrayList<Entity> isColliding(Rectangle rect)	Check whether the entity's hitbox is colliding with the remaining entities in the game

+ boolean isCollidingWithPlayer(Entity entity)	Check whether the entity is colliding with the player
+ void add(Entity e)	Add the entity to the entity list either straightforward or through queue
+ void remove(Entity e)	Remove the entity from the entity list either straightforward or through queue
+ CollisionManager getInstance()	Singleton pattern

5. package core.game.effect

5.1. class GreenForceWave

This class represents a force wave effect which travels in a straight line at a certain angle.

5.1.1. Fields

- *float MOVEMENT_SPEED*	Set the value to 6.f * Tile.SIZE
- AnimatedSprite sprite	The graphic representation of the green force wave
- Rectangle bound	The hitbox of the wave
- Vector2f position, direction	Position and direction of the wave
- int size	Size of the wave
- float angle	Angle that the wave travels

5.1.2. Constructor

+ GreenForceWave(Vector2f position, int size, float angle)	Initialize all fields and set the sprite state to IDLE

5.1.3. Methods

+ void update(float deltaTime)	Update the position of the wave
+ void render()	Render the sprite
+ Rectangle getBound()	Return the hitbox

5.2. class HealthEffect extends GameBehavior

This class represents a text effect whenever the character has been healed or damaged.

5.2.1. Fields

- Vector2f position	Position of the text
- float amount	Amount of health that has been gained or lost
- float time, maxTime	Represent the duration that the text has appeared and the span of time that the text will stay visible

5.2.2. Constructor

+ HealthEffect(Vector2f position, float amount)	Initialize all fields

5.2.3. Methods

+ void update()	Update the text's position and check whether the time has reached its limit
+ void render()	Render the text

6. package core.game.entity

6.1. interface HostileEntity

This interface is used to mark that a certain entity is a hostile one.

6.2. abstract class Entity extends GameBehavior

This class is a base class for all entities in the game. It contains movement methods and other necessary methods in the game and also checks whether an entity is collided with tiles or not.

6.2.1. Fields

- String id	The ID representing this entity
# Vector2f position	The entity's position
# Vector2f direction	The direction that the entity is heading
# float movementSpeed	The entity's movement speed
# float maxHealth, health	Max health and current health of this entity
# float attackDamage, attackCooldownTime,	The entity's attack damage, attack cooldown time and latest

lastAttackTime	attack time
# Rectangle bound	The hitbox of this entity
# ArrayList<Entity> collidingEntity	A list storing other entities that are colliding with this entity
- boolean[] movable	Initialize to an array of size 4. (Top, Right, Bottom, Left)

6.2.2. Constructor

+ Entity(String id, int layer)	Initialize all fields and add this entity to the entity list

6.2.3. Methods

# *void onDeath()*	An abstract method to handle each entity's behavior when its health runs out
# *void onTakingDamage()*	An abstract method to handle each entity's behavior when it takes damage
# void delete()	When this entity's health runs out, remove the entity from all lists that it resides and plays an audio related to it
# void drawBound()	Draw the entity's hitbox
# void drawHealthBar()	Draw the entity's health bar
+ void takeDamage(float damage)	Reduce the entity's health by the parameter's amount and display a health effect and an audio related to it

+ void heal(float heal)	Heal the entity's health by the parameter's amount but not exceeding its max health and display a health effect
# void attack(Entity e)	Attack another entity if the cooldown time is already expired
# void move(Vector2f direction)	Check the possibility of moving the entity towards a certain direction and update its position
- void checkCollision(Vector2f dv)	Check the collision state of this entity towards all the direction
+ void setHealth(float health)	If the entity's health is below or equal to 0, set it to 0 and call onDeath() to handle it. Otherwise, set it to match the parameter but cannot exceed its max health
+ void setMovementSpeed(float speed)	Set the movementSpeed to be equal to the parameter's value multiply by the size of the tile
+ float getMovementSpeed()	Return the movementSpeed divided by the size of the tile
Getter/Setter for all the remaining fields	

6.3. class Player extends Entity

This class represents the player's playing character. The character's movement is controlled by the keyboard input WASD.

6.3.1. Fields

- Vector2f direction	The direction that the player is heading towards
- AnimatedSprite sprite	The graphic representation of the player's character
- Item item	The player's weapon of choice
- int blinkPeriod, maxBlink, blinkTime	The blink period, max blink and blink time of the character

6.3.2. Constructor

+ Player()	Initialize all fields

6.3.3. Methods

+ void update()	Update player's behavior
+ void render()	Render player's character, item and health bar
# void onTakingDamage()	Create a blink animation to the character
# void onDeath()	Set the Game Over state
- void calculateDirection()	Move the player's character according to the player's keyboard input
+ Item getItem()	Return the character's item

7. package core.game.entity.enemy

7.1. class EnemySpawner implements Runnable

This class is used to spawn the player's enemies.

7.1.1. Fields

- Random random	Random the value to decide which enemy types to spawn next
- float NEXT_SPAWN_TIME	The duration in which to spawn the next surge of enemies, set to 5.f
- float time	Current time since the latest spawn event
- boolean running	The game current status

7.1.2. Constructor

+ EnemySpawner()	Random a value

7.1.3. Methods

+ void run()	State that the game is running, spawn enemies according to the game's scheme
- void spawnEnemy()	Spawn enemies and adjust the number and spawn rate to match the game time. Also spawn different enemy types according to their spawn percentage rate
+ void stop()	Set running to false, pause the game

7.2. class Imp extends Entity implements HostileEntity

This class represents an explosive type of hostile entity known as “Imp”. It triggers a fuse whenever the player is in its certain range. It can only deal damage to the player through explosion.

7.2.1. Fields

- AnimatedSprite sprite, explosionSprite	The graphic representation of the Imp, in its default state and explosion state respectively
- float baseDetonateTime, detonateTime, time	The base detonate time, current detonate time according to player’s level and duration that the player is in its explosion range respectively
- float baseExplosionRange, explosionRange	The explosion range of an Imp
- boolean isDetonated	State whether an Imp has already exploded or not
- float blinkTime, blinkStartFrequency	The current blink duration and blink frequency
- boolean showBlink, lastShowBlink	State whether the current blink effect will be displayed or not and the latest shown blink effect

7.2.2. Constructor

+ Imp(Vector2f spawn)	Initialize all fields

7.2.3. Methods

+ void update()	Update the Imp status according to player's position, level and time
+ void render()	Render the Imp's sprite according to its status
# void onDeath()	Remove this Imp from the game and spawn a coin on its place

7.3. class Undead extends Entity implements HostileEntity

This class represents a normal type of hostile entity known as "Undead". It attacks the player when it gets close enough to be able to.

7.3.1. Fields

- AnimatedSprite sprite	The graphic representation of the Undead

7.3.2. Constructor

+ Undead(Vector2f spawn)	Initialize all fields

7.3.3. Methods

+ void update()	Update the Undead's position and check whether it can attack the player
+ void render()	Render the Undead's sprite and its

	health bar
# void onDeath()	Remove this Imp from the game and spawn a coin on its place

8. package core.game.item

8.1. abstract class Item

This class is a base class for all the items in the game. It provides general information to each of the items.

8.1.1. Fields

# String NAME, SPRITE_PATH	The name and the sprite path of the item
# float BASE_ATTACK_DAMAGE, BASE_ATTACK_COOLDOWN_TIME	Base attack damage and base attack cooldown time of the item
# float attackDamage, attackCooldownTime, attackTime	Attack damage, attack cooldown time and attack time of the item
# Vector2f position, direction	Position and direction that the item is located and facing
# int level	Level of the item

8.1.2. Constructor

+ Item(String name, String spritePath, float baseAttackDamage, float baseAttackCooldownTime)	Initialize all fields and set the level to 1

8.1.3. Methods

+ *void update(float deltaTime)*	An abstract method for updating the item
+ *void render()*	An abstract method for rendering the item
+ void reset()	Reset the item's level to 1
+ *void attack()*	An abstract method for setting the item's attacking behavior
+ *void setPosition(Vector2f position)*	An abstract method for setting the item's position
+ *void setDirection(Vector2f direction)*	An abstract method for setting the item's direction
+ void increaseLevel()	Increment the item's level by 1
+ String getName()	Return the item's name in its format
Getter for attackDamage, attackCooldownTime, level, SPRITE_PATH	

8.2. class MagicWand extends Item

This class represents SpellCaster's weapon "Magic Wand". It creates a force wave that deals damage to the nearest enemy.

8.2.1. Fields

- Sprite sprite	The graphic representation of the Magic Wand

- AnimatedSprite hitFxSprite	The graphic representation of the Magic Wand's hit animation
- Vector2f position	The Magic Wand's position
- ArrayList<GreenForceWave> forceWaveLists	The list of all the force waves that are currently active
- float BASE_ATTACK_RANGE	The base attack range of the Magic Wand
- float attackRange	The current attack range of the Magic Wand

8.2.2. Constructor

+ MagicWand()	Initialize all fields

8.2.3. Methods

+ void update(float deltaTime)	Update all the force waves that are currently in the game. It checks whether each wave does hit an enemy or not or whether it travels out of the rendering window. The waves that satisfy either of the conditions will later be deleted
+ void render()	Render the force wave's animation
+ void reset()	Clear the forceWaveLists and reset the Magic Wand's level
+ void attack()	Adjust the fields according to player's level and check whether there is an enemy in its attack range

+ void setPosition(Vector2f position)	Return the Magic Wand's position

8.3. class RustySword extends Item

This class represents Knight's weapon "Rusty Sword". It is a sword that comes with enormous power and damage. The player, however, has to pay the price for its bulky size by the player's agility.

8.3.1. Fields

- Sprite sprite	The graphic representation of the Rusty Sword
- AnimatedSprite hitFxSprite	The graphic representation of the Rusty Sword's hit animation
- Hitbox[] hitboxes	The hitbox of the Rusty Sword
- Vector2f position	The Rusty Sword's position
- int direction	The direction that the Rusty Sword is facing towards

8.3.2. Constructor

+ RustySword()	Initialize all fields

8.3.3. Methods

+ void update(float deltaTime)	Update the Rusty Sword's status
+ void render()	Render the Rusty Sword's

	animation
+ void attack()	Adjust the fields according to player's level and check whether there is an enemy in its attack range
+ void setPosition(Vector2f position)	Set the Rusty Sword's position
+ void setDirection(Vector2f direction)	Set the Rusty Sword's direction

8.4. class ShortSword extends Item

This class represents Braves' weapon "Short Sword". It is a sword that comes with light-speed agility. Sadly, the power is what it is lacking.

8.4.1. Fields

- Sprite sprite	The graphic representation of the Short Sword
- AnimatedSprite hitFxSprite	The graphic representation of the Short Sword's hit animation
- Hitbox[] hitboxes	The hitbox of the Short Sword
- Vector2f position	The Short Sword's position
- int direction	The direction that the Short Sword is facing towards

8.4.2. Constructor

+ ShortSword()	Initialize all fields

8.4.3. Methods

+ void update(float deltaTime)	Update the Short Sword's status
+ void render()	Render the Short Sword's animation
+ void attack()	Adjust the fields according to player's level and check whether there is an enemy in its attack range
+ void setPosition(Vector2f position)	Set the Short Sword's position
+ void setDirection(Vector2f direction)	Set the Short Sword's direction

9. package core.game.world

9.1. class Coin extends GameBehavior

This class represents Coin that increase the player's experience point when collected. Coin will be automatically attracted towards the player when it is in a certain range.

9.1.1. Fields

- Vector2f position	The coin's position
- Rectangle bound	The coin's hitbox
- AnimatedSprite sprite	The graphic representation of the

	coin
- float time	The coin’s position

9.1.2. Constructor

+ Coin(Vector2f position)	Initialize all fields

9.1.3. Methods

+ void update()	Update the state of the coin when the player gets in its radius. Have an effect on the player’s health and experience point
+ void render()	Render the coin’s animation
+ Rectangle getBound()	Return the coin’s hitbox

9.2. class Tile extends GameBehavior

This class represents the tile in the map. It holds the tile’s information and coordinates.

9.2.1. Fields

+ <u>int SIZE</u>	Set the value to 32
- Sprite sprite	The graphic representation of the tile, set the default value to null
- boolean solid, transparent	Set both value to false
- int x, y	The coordinate of the tile

9.2.2. Constructor

+ Tile(Sprite sprite, int x, int y)	Call another constructor with a specific value
+ Tile(Sprite sprite, int x, int y, int layer)	Initialize all fields

9.2.3. Methods

+ void setTransparent()	Set the tile to be transparent
+ void render()	Render the tile's sprite
+ boolean isSolid()	Return whether the tile is solid or not
+ boolean isEmptyTile()	Return whether the tile is null or not

9.3. class World

This class represents the world that the game occurred in. It holds all the information including the tile's list and its collision state.

9.3.1. Fields

- <u>World instance</u>	Singleton pattern
+ Vector2f spawnPoint	Indicate the spawn point
- int mapWidth, mapHeight, mapMaxLayer	The map width, map height and map max layer

- Tile[][][] tileMaps	All the tiles in the game
- boolean[][] collisionMaps	The collision on the map
- ArrayList<Coin> coins	The list of all coins in the world
- ArrayList<Entity> enemyLists	The list of all enemies in the world
- Player player	The player's character
- Thread enemySpawnerThread	The thread that manages the enemy spawning event
- EnemySpawner enemySpawner	Manage the enemy spawning event

9.3.2. Methods

+ void initialize()	Initialize all fields
+ World getInstance()	Singleton pattern
+ void spawnCoin(Vector2f pos)	Add the coin in the parameter's position
+ void addEnemy(Entity e)	Add an enemy to the list
+ void removeEnemy(Entity e)	Remove an enemy from the list
+ boolean isPosSolidTile(float x, float y)	Check whether the given position is a solid tile or not
+ boolean isCoordSolidTile(int x, int y)	Check whether the given coordinate is a solid tile or not
+ Vector2f posToCoord(float x, float y)	Position to coordinate conversion
+ Vector2f coordToPos(int x, int y)	Coordinate to position conversion

+ Tile getTile(float x, float y, int layer)	Return the tile according to the parameters' values
+ Tile getTile(Vector2f pos, int layer)	Return the tile according to the parameters' values
Getter for player, enemyLists, coins, spawnPoint, mapWidth, mapHeight and mapMaxLayer	

9.4. class WorldLoader

This class is used to load all tiles in the world.

9.4.1. Fields

- Tile[][][] tileMaps	All the tiles in the game
- boolean[][] collisionMaps	The collision on the map
- Vector2f spawnPoint	Indicate the spawn point

9.4.2. Methods

+ void load(SpriteSheet tileset, String... paths)	Load the map and initialize all the properties on the map
- boolean isSolid(Color c)	State whether a pixel is solid or not
Getter for all fields	

10. package core.inputHandler

10.1. class KeyboardHandler implements EventHandler<KeyEvent>

This class is used to handle the player's keyboard inputs.

10.1.1. Fields

- Set<KeyCode> keydowns, lastKeydowns	The set of currently pressed keys and the set of latest pressed keys respectively

10.1.2. Methods

+ void initialize()	Initialize all fields
+ handle(KeyEvent e)	Handle the keyboard input events
+ boolean onKeyPressed(KeyCode k)	Return whether a certain key is currently being pressed or not
+ boolean isKeyPressed(KeyCode k)	Return whether a certain key is pressed or not

10.2. class MouseHandler implements EventHandler<MouseEvent>

This class is used to handle the player's mouse inputs.

10.2.1. Fields

- Set<String> mousedowns	The set of currently clicked button name
- Vector2f mousePosition	The cursor's position
- boolean lastMouseDown	State whether the mouse is

	currently clicked or not

10.2.2. Methods

+ void initialize()	Initialize all fields and set lastMouseDown to false
+ handle(MouseEvent e)	Handle the mouse input events
+ boolean isMouseDown(MouseButton mb)	Return whether a certain button is clicked or not
+ Vector2f getMousePosition()	Return the cursor's position

11. package core.sprite

11.1. class Sprite implements Comparable<Sprite>

This class is used to create sprites from both the system resources and from the sprite sheet.

11.1.1. Fields

- int x, y, size	The x, y value and size of the sprite
- Image image	The image located by the sprite path
- SpriteSheet sheet	The spritesheet

11.1.2. Constructor

+ Sprite(String path)	Load the image through SpriteLoader

+ Sprite(SpriteSheet sheet, int x, int y)	Initialize fields according to parameters' values and load the SpriteSheet

11.1.3. Methods

+ void loadSpriteSheet()	Load the image through PixelReader
+ Image getImage()	Return the image
+ int getSize()	Return size
+ int getWidth()	Return the width of the image
+ int getHeight()	Return the height of the image
+ int compareTo(Sprite o)	Compare the hash code of this object to another one

11.2. class SpriteLoader

This class is used to load the images from the system resource folders.

11.2.1. Methods

+ Image load(String src)	Return an image loaded through ClassLoader

11.3. class SpriteSheet

This class is used to load the sprite sheet from the system resource folders and maps the color sheets to the sprite sheets.

11.3.1. Fields

+ SpriteSheet tileset	Initialize the default SpriteSheet
- int width, height, size	The width, height and size of the image
- Image image	The image located by the sprite sheet path
- Map<Integer, Sprite> colorMaps	Set the default value to null

11.3.2. Constructor

+ SpriteSheet(String spriteSheetpath, String colorSheetPath, int size)	Load the image through SpriteLoader and load the color sheet. Also initialize all fields

11.3.3. Methods

- void loadColorSheets(String path)	Load the color sheet through SpriteLoader and put each pixel value into the color map
Getter for all fields except tileset	

12. package core.sprite.animation

12.1. class AnimatedSprite

This class is used to play an animation on the sprite by using the state to manage the animation.

12.1.1. Fields

- int frameWidth, frameHeight	The frame's width and height respectively
- int frame, frameCol	Number of frames and frame column count
- float timePerFrame, time	Duration that each specific frame will be shown and current time respectively
- Sprite sprite	The graphic representation
- boolean reverse	Set the default value to false
- Vector2f offset	Screen offset
- State currentState	Current state of the sprite
- Map<State, AnimationState> animationState	Store the state that maps to the animation state
- Runnable event	A runnable object

12.1.2. Constructor

+ AnimatedSprite(String path, int frameRow, int frameCol, int frameWidth, int frameHeight, int frameCount)	Initialize all fields and set the state to IDLE

12.1.3. Methods

+ void setStateData(State state, State nextState, int start, int end)	Set the next state value according to present time

+ void update(float deltaTime)	Update the frame
+ void render(int x, int y, int w, int h, float angle)	Render the sprite on the screen
+ Vector2f getOffset()	Return the offset value
+ int getWidth()	Return the sprite's width
+ int getHeight()	Return the sprite's height
+ void setEventHandler(Runnable event)	Set the event to match the parameter
Setter for state, reverse, offset and timePerFrame	

12.2. class AnimationState

This class is used to hold the current and next state and start and stop frame index.

12.2.1. Fields

+ enum State	Holds the following states: IDLE, PLAY
- State state	Current state
- State nextStage	Next stage
- int startFrame, endFrame	The start and the end frame

12.2.2. Constructor

+ AnimationState(State state, State nextStage, int startFrame, int	Initialize all fields

endFrame)	

12.2.3. Methods

Getter for all fields	

13. package core.ui.component

13.1. abstract class UIComponent

This class is a base class for all UI-involving classes in the game. It provides graphics context and updating method for the class that inherits it.

13.1.1. Fields

# GraphicsContext gc	A graphics context for the game

13.1.2. Constructor

+ UIComponent()	Initialize the graphics context

13.1.3. Methods

+ *void update(float deltaTime)*	An abstract method for updating the UI.

13.2. class Button extends UIComponent

This class represents a button UI for the game.

13.2.1. Fields

- Label label, rightHoverCursor, leftHoverCursor	Labels containing the button's information
- SubWindow bound	Window boundary of the button
- Color hoverColor	The color when the mouse is hovering
- boolean isHover	State whether the mouse is hovering or not
- Runnable event	A runnable event

13.2.2. Constructor

+ Button(String text, int x, int y, int w, int h)	Initialize all fields and adjust the button to match the settings correctly

13.2.3. Methods

+ void update(float deltaTime)	Update the UI when the player hovers the cursor over the button
+ void setOnClick(Runnable event)	Set the event to match the parameter's value
+ void setHoverColor(Color hoverColor)	Set the hover color to match the parameter's value
+ SubWindow getBound()	Return the bounded button
+ Label getLabel()	Return the label

13.3. class Image extends UIComponent

This class represents the image UI, which can be resizable.

13.3.1. Fields

- SubWindow bound	Window boundary of the image
- Sprite sprite	The graphic representation of the image
- float scale	The image scaling

13.3.2. Constructor

+ Image(String path, int x, int y, int size)	Initialize all fields

13.3.3. Methods

+ void update(float deltaTime)	Draw the image on the graphics context
+ SubWindow getBound()	Return the bounded image
+ Sprite getSprite()	Return the sprite

13.4. class Label extends UIComponent

This class represents a text UI in the game.

13.4.1. Fields

- Position pos	The position of the text in the label
- Font font	The text's font style
- Color color	The text's color
- String text	The text itself
- int x, y	The x and y values of the text
- Color shadowColor	The text's shadow color
- boolean isTextShadow	State whether the text is shadowed or not
- int shadowOffset	The text's shadow offset
- Text fxText	The effect text

13.4.2. Constructor

+ Label(String text, int x, int y)	Initialize all fields

13.4.3. Methods

+ void update(float deltaTime)	Update the text UI on the graphics context
+ void setFont(Font font)	Set the font and fxText's font to match the parameter's value
+ void setFontSize(double size)	Set the font size
+ int getTextWidth()	Return the fxText's width
+ int getTextHeight()	Return the fxText's height

+ void setText(String text)	Set the text and fxText to match the parameter's value
Setter for pos, isTextShadow, color, shadowColor and shadowOffset	

13.5. class ProgressBar extends UIComponent

This class represents the progress bar UI in the game.

13.5.1. Fields

- float maxProgress, progress	Max progress and player's current progress
- Rectangle bound	A rectangle that stores the progress bar
- Color backgroundColor, foregroundColor	Progress bar's background color and foreground color
- boolean isReverse	State whether the progress will be reversed or not
- Color borderColor	Progress bar's border color
- int borderSize, borderRadius	Progress bar's border size and border radius

13.5.2. Constructor

+ ProgressBar(int x, int y, int w, int h, float maxProgress)	Initialize all fields

13.5.3. Methods

+ void update(float deltaTime)	Update the color that represents a change in progress on the graphics context
+ void setProgress(float progress)	Set the progress according to the parameter's value, but not more than max progress value
+ float getProgress()	Return the player's progress
Setter for all fields except bound	

13.6. class SubWindow extends UIComponent

This class represents the blank rectangle window with rounded corners.

13.6.1. Fields

- Rectangle bound	A rectangle that represents a window
- Color borderColor, backgroundColor	Window's border color and background color
- int borderSize, borderRadius	Window's border size and border radius

13.6.2. Constructor

+ SubWindow(int x, int y, int w, int h)	Initialize all fields

13.6.3. Methods

+ void update(float deltaTime)	Update the window on the graphics context
+ Rectangle getBound()	Return the bounded window
Setter for all fields except bound	

13.7. enum Position

It is used to hold the following positions:
LEFT, CENTER, RIGHT

14. package core.ui

14.1. class GameOverWindow extends UIComponent

This class represents the gameover overlay and also includes a button which links the player back to the main menu.

14.1.1. Fields

- Color backgroundColor	The window's background color
- Label text	A text displayed to the player
- Button quit	A button which links the player back to the main menu

14.1.2. Constructor

+ GameOverWindow()	Initialize all fields and adjust the text and button to display the correct information

14.1.3. Methods

+ void update(float deltaTime)	Update the window

14.2. class PauseWindow extends UIComponent

This class represents the pause window overlay and also includes a button which links the player back to the game and another button which links the player to the main menu, exiting the current game.

14.2.1. Fields

- Color backgroundColor	The window's background color
- Label text	A text displayed to the player
- Button quit, back	Quit game and back to game button

14.2.2. Constructor

+ PauseWindow()	Initialize all fields and adjust the text and buttons to display the correct information

14.2.3. Methods

+ void update(float deltaTime)	Update the window

14.3. class StatusWindow extends UIComponent

This class represents the status window overlay which shows the player's status and the character's item status. It also contains another class that displays each stat's information.

14.3.1. class StatItem

14.3.1.1. Fields

- Label item, item2	Display the stat's name and its current value

14.3.1.2. Constructor

+ StatItem(int x, int y)	Initialize all fields and adjust the status to display the correct information

14.3.1.3. Methods

+ void update(String name, float amount, float deltaTime)	Update the stat according to its format

14.3.2. Fields

- SubWindow window	The status window

- Label playerTitle, itemTitle	Display the player and item label
- StatItem movementSpeed, maxHealth, attackDamage, attackSpeed, weaponLevel	The status of player and item to be displayed on the window

14.3.3. Constructor

+ StatusWindow()	Initialize all fields and adjust the status to display the correct information

14.3.4. Methods

+ void update(float deltaTime)	Update each StatItem to display its current state

14.4. class UpgradeWindow extends UIComponent

This class represents the upgrade window overlay which shows when the player's level increases. The player can select abilities to upgrade and then automatically return back to the game.

14.4.1. class Coin

14.4.1.1. Fields

+ int x, y, size, frame	Properties of a coin

14.4.1.2. Constructor

+ Coin()	Initialize all fields using Random

14.4.2. class ItemData

14.4.2.1. Fields

+ String name, description, path	Information of the item
+ Runnable action	The item’s action

14.4.2.2. Constructor

+ ItemData(String name, String description, String path, Runnable action)	Initialize all fields

14.4.3. class UpgradeItem

14.4.3.1. Fields

+ Button button	Upgrade button
+ Label label, label2	Name and description of the upgraded item
- Image img	Image of the item

14.4.3.2. Constructor

+ UpgradeItem(ItemData item, int x, int y, int w, int h)	Initialize all fields and adjust all settings to display the

	correct information

14.4.3.3. Methods

+ void update(float deltaTime)	Update the UI

14.4.4. Fields

- int COIN_SIZE	Set the value to 500
- int GRAVITY	Set the value to WINDOW_HEIGHT / 2
- float COIN_FRAME_TIME	Set the value to 0.15f
- Color backgroundColor	The window’s background color
- Sprite coin	The graphic representation of the coin
- ArrayList<Coin> coins	A list of coins in the background
- float time	Set the default value to 0.f
- SubWindow window	The displayed window
- Label levelUp	A label showing that the player has leveled up
- ArrayList<UpgradeItem> upgradeItemLists	A list of the upgraded item UIs

14.4.5. Constructor

+ UpgradeWindow()	Initialize all fields and adjust the

	text to display the correct information

14.4.6. Methods

+ void update(float deltaTime)	Update the UI
- void drawBackground(float deltaTime)	Draw the coin falling background animation
+ void addItems()	Add newly upgraded item to the list

15. package util

15.1. class ColorUtil

This class is used for color parsing.

15.1.1. Methods

+ int parseColorToInt(int r, int g, int b)	Returns color code from RGB (Red, Green, Blue)
+ int parseColorToInt(int r, int g, int b, int a)	Returns color code from RGBA (Red, Green, Blue, Alpha)
+ int parseColorToInt(Color c)	Returns a color code of a certain color
+ Color parseRGBToColor(int r, int g, int b)	Returns color from a certain RGB value

15.2. class Time

This class is used to access present time.

15.2.1. Methods

+ float getNanoSecond()	Returns time in nanoseconds

15.3. class Vector2f

This class is used to calculate vector's arithmetic.

15.3.1. Fields

+ float x, y	The x-axis and y-axis value of a 2D-float vector

15.3.2. Constructor

+ Vector2f()	Initialize a 2D vector with (0,0)
+ Vector2f(Vector2f vec)	Copy constructor
+ Vector2f(float x, float y)	Initialize a 2D vector with (x,y)
+ Vector2f(float angle)	Initialize a 2D vector with a certain angle

15.3.3. Methods

+ Vector2f addEqual(Vector2f other)	Add this vector's value by another vector's value
+ Vector2f add(Vector2f other)	Returns a new vector with its value coming from the sum of this vector

	and another vector
+ Vector2f add(float other)	Returns a new vector with its value coming from the sum of this vector and a float value
+ Vector2f subtractEqual(Vector2f other)	Subtract this vector's value by another vector's value
+ Vector2f subtract(Vector2f other)	Returns a new vector with its value coming from the difference between this vector and another vector
+ Vector2f subtract(float other)	Returns a new vector with its value coming from the difference between this vector and a float value
+ Vector2f multiplyEqual(float other)	Multiply this vector's value by a float value
+ Vector2f multiply(float other)	Returns a new vector with its value coming from the product of this vector and a float value
+ float getSize()	Returns the size of this vector
+ Vector2f normalize()	Normalize and return this vector
+ void setZero()	Set the value of both x and y to 0
+ Vector2f round()	Round the value of both x and y
+ String toString()	Returns the string of the value x and y of a vector in the desired format
+ int compareTo(Vector2f other)	Compare the size of this vector to another one
+ boolean isZero()	Returns true if the value of both x and y is equal to 0, otherwise

	returns false
+ boolean isInNorth(Vector2f other)	Returns true if ‘other’ vector is lying in the north of this vector
+ boolean isInSouth(Vector2f other)	Returns true if ‘other’ vector is lying in the south of this vector
+ boolean isInWest(Vector2f other)	Returns true if ‘other’ vector is lying in the west of this vector
+ boolean isInEast(Vector2f other)	Returns true if ‘other’ vector is lying in the east of this vector
+ boolean[] getDirection(Vector2f other)	Returns an array with each index indicating whether ‘other’ vector is lying in the north, south, east, west of this vector respectively
+ boolean[] toDirection()	Returns an array with each index indicating the direction in which the vector is pointing towards (in the order of south, east, north, west)

16. package scene

16.1. abstract class BaseScene

This class is a base class for all scenes in this game.

16.1.1. Fields

# Stage stage	The main stage of the game
# GraphicsContext gc	The game’s graphics context
- String ID	This scene’s ID

16.1.2. Constructor

+ BaseScene(String ID, Stage stage)	Initialize all fields

16.1.3. Methods

+ *void onLoadScene()*	An abstract method to handle each scene's event properly
+ *void update(float deltaTime)*	An abstract method for updating the scene.
+ String getID()	Return the scene's ID

16.2. class GameScene extends BaseScene

This class is used to update and render the current game state and all the game behavior. It also initializes behavior manager , collision manager, world and camera.

16.2.1. Fields

- Color backgroundColor	The scene's background color
- ProgressBar expBar	The player's progression bar
- Label level, timer	Player's level and game timer
- float lastTime	The latest update time
- UpgradeWindow upgradeWindow	The game's upgrade window
- StatusWindow statusWindow	The game's status window
- GameOverWindow	The game's game over window

gameOverWindow	
- PauseWindow pauseWindow	The game's pause window

16.2.2. Constructor

+ GameScene(String ID, Stage stage)	Initialize all fields and adjust them to match the settings correctly

16.2.3. Methods

+ void onLoadScene()	Initialize managers, world and camera and plays the battle background music
+ void update(float deltaTime)	Update player's level and game state. Also render the screen according to the game state and render the UI to match their values

16.3. class MainMenuScene extends BaseScene

This class represents the main menu scene of the game. There is a start button for the player to start playing the game.

16.3.1. Fields

- Label title	The title of the game
- Sprite background	The main menu's background
- Button startButton, exitButton	A button to start the game and another one to exit the game

16.3.2. Constructor

+ MainMenuScene(String ID, Stage stage)	Initialize all fields and adjust them to match the settings correctly

16.3.3. Methods

+ void onLoadScene()	Play the background music
+ void update(float deltaTime)	Update the scene

16.4. class SelectingCharacterScene extends BaseScene

This class provides a page for the player to select the desired characters in the game. It also contains another class that is used to create a button to select each character properly.

16.4.1. class CharacterButton

16.4.1.1. Fields

+ Button button	The button for selecting a character
+ Label label	Character’s information
- Image img	Character’s image

16.4.1.2. Constructor

+ CharacterButton(String name, String path, PlayerCharacter character, int x, int y, int w, int h)	Initialize all fields and adjust them to display the information correctly

16.4.1.3. Methods

+ void update(float deltaTime)	Update the fields

16.4.2. Fields

- Label title	Display a text to indicate the player to select a character
- Sprite background	Display the main menu background
- SubWindow window	A window to select a character
- ArrayList<CharacterButton> characterButtonLists	A list that contains character selection buttons

16.4.3. Constructor

+ SelectCharacterScene(String ID, Stage stage)	Initialize all fields and adjust them to match the settings correctly

16.4.4. Methods

+ void onLoadScene()	Do nothing

+ void update(float deltaTime)	Update the scene

16.5. class StartingScene extends BaseScene

This class represents the game's starting scene when the game has been opened. It is used to display the game's studio name.

16.5.1. Fields

- Label title	The studio's title
- float time, maxTime	The current time since this scene has been displayed and the total duration that this scene will be shown

16.5.2. Constructor

+ StartingScene(String ID, Stage stage)	Initialize all fields and adjust them to match the settings correctly

16.5.3. Methods

+ void onLoadScene()	Do nothing
+ void update(float deltaTime)	Update the scene

17. package logic

17.1. class GameLogic

This class is used to hold the game state and game scene. It initializes the game loop and holds the player's experience point and level and also holds the time counter as well.

17.1.1. Fields

- Window window	The game's window
- GraphicsContext gc	The game's graphics context
- ArrayList<BaseScene> sceneLists	A list of all scenes in the game
- int currentScene	Game's current scene
- Font defaultFont	Game's default font
- GameState gameState	Current game state
- Map<GameState, Runnable> gameStateEvents	A map for the game state
- float maxExperience, experience	Player's max experience and current experience point
- int level	Player's level
- int timeCounter	Game's time counter
- PlayerCharacter character	Player's selected character
- <u>GameLogic instance</u>	Singleton pattern

17.1.2. Methods

+ <u>GameLogic getInstance()</u>	Singleton pattern

+ void initialize(Stage stage)	Initialize fields, game state and game loop
+ void initializeGameState()	Set the game state according to action event and initialize some fields to default value
+ void initializeGameLoop()	Use AnimationTimer to keep track of the game progress over time. Also showing the FPS to the player
+ void update(float deltaTime)	Clear screen and update the next scene
+ void setCurrentScene(int index)	Set the scene to be shown
+ void nextScene()	Set the next scene
+ void nextLevel()	Adjust the player's experience, max experience point and the game state
+ void incrementLevel()	Increase player's level by 1
+ void setGameState(GameState gameState)	Run the current game state
+ void setOnGameStateChangeTo(GameState gameState, Runnable e)	Put the game state to the gameStateEvents list
+ String timeCounterToString()	Return current time in the desired format
Getter for gc, defaultFont, maxExperience, experience, level, gameState, character and timeCounter	
Setter for maxExperience, experience, character and timeCounter	

17.2. class Window

This class is used to create JavaFX window and canvas and get Graphics Context from the canvas. It also initializes both keyboard and mouse input handler.

17.2.1. Fields

+ int WINDOW_WIDTH	Set the value to 1024
+ int WINDOW_HEIGHT	Set the value to WINDOW_WIDTH / 16 * 9
- Stage stage	The game's main stage
- Scene scene	The current scene
- GraphicsContext gc	The game's graphics context

17.2.2. Constructor

+ Window(Stage stage)	Initialize the stage, window and input handlers

17.2.3. Methods

- void initializeWindow()	Set the stage properties and close request. Also initialize the canvas and set it on the stage
- void initializeKeyboardHandler()	Initialize the keyboard input handler
- void initializeMouseHandler()	Initialize the mouse input handler

+ void setTitle(String text)	Set the stage's title
Getter for all fields	

17.3. enum GameState

It is used to hold the game's following states: PLAY, PAUSE, EXIT, UPGRADE, GAME_OVER

17.4. enum PlayerCharacter

It is used to hold the game's following playable characters: BRAVES, KNIGHT, SPELLCASTER. It also holds each character's sprite, default weapon and default movement speed.

17.4.1. Fields

- String spritePath	The character's sprite path
- Item item	The character's weapon
- float movementSpeed	The character's movement speed

17.4.2. Constructor

+ PlayerCharacter(String spritePath, Item item, float movementSpeed)	Initialize all fields

17.4.3. Methods

Getter for all fields	

17.5. class Main extends Application

Initialize the stage by using GameLogic.getInstance().initialize(stage) then launch the application.

Note that ‘javafx.web’ needs to be included in the VM Arguments in order to use the MediaPlayer