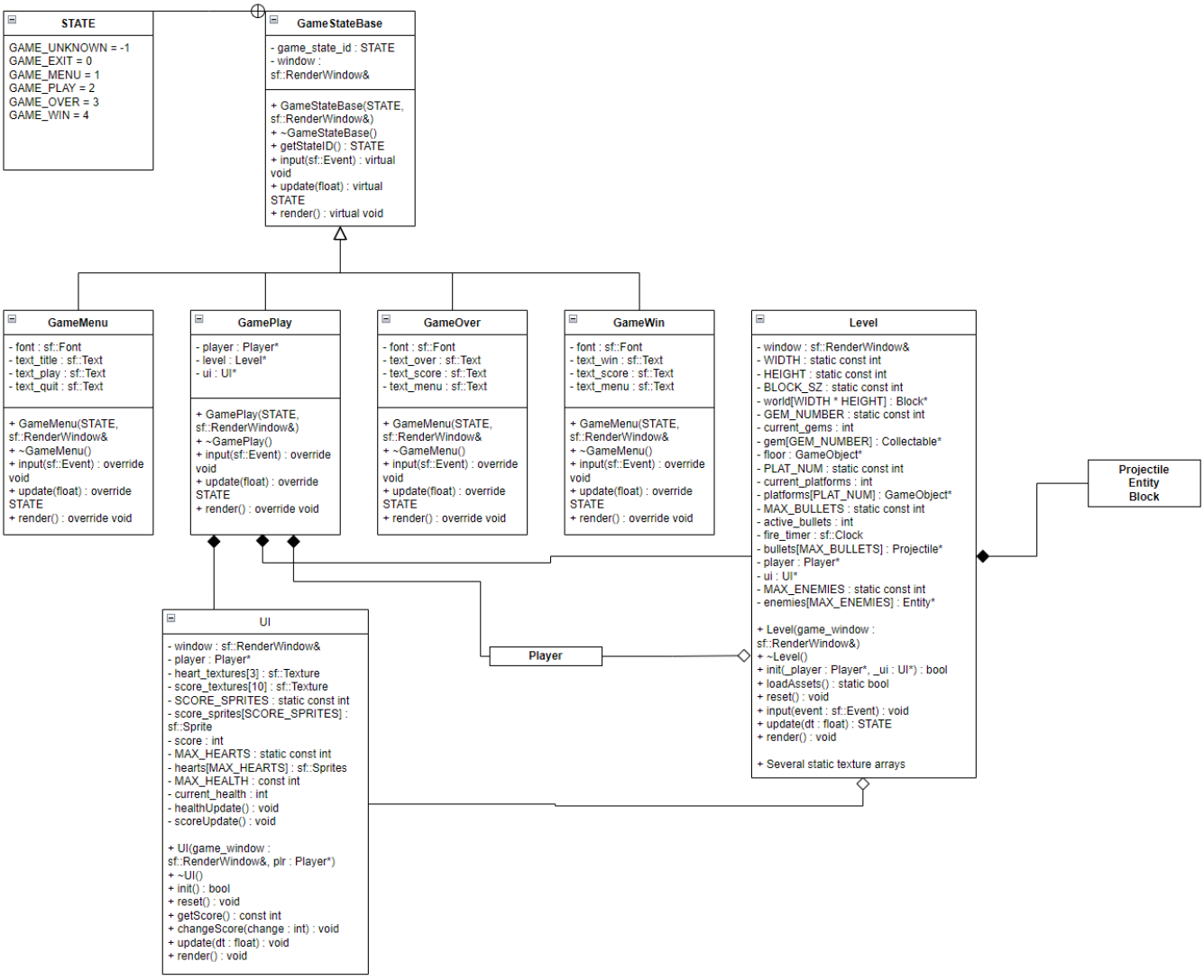
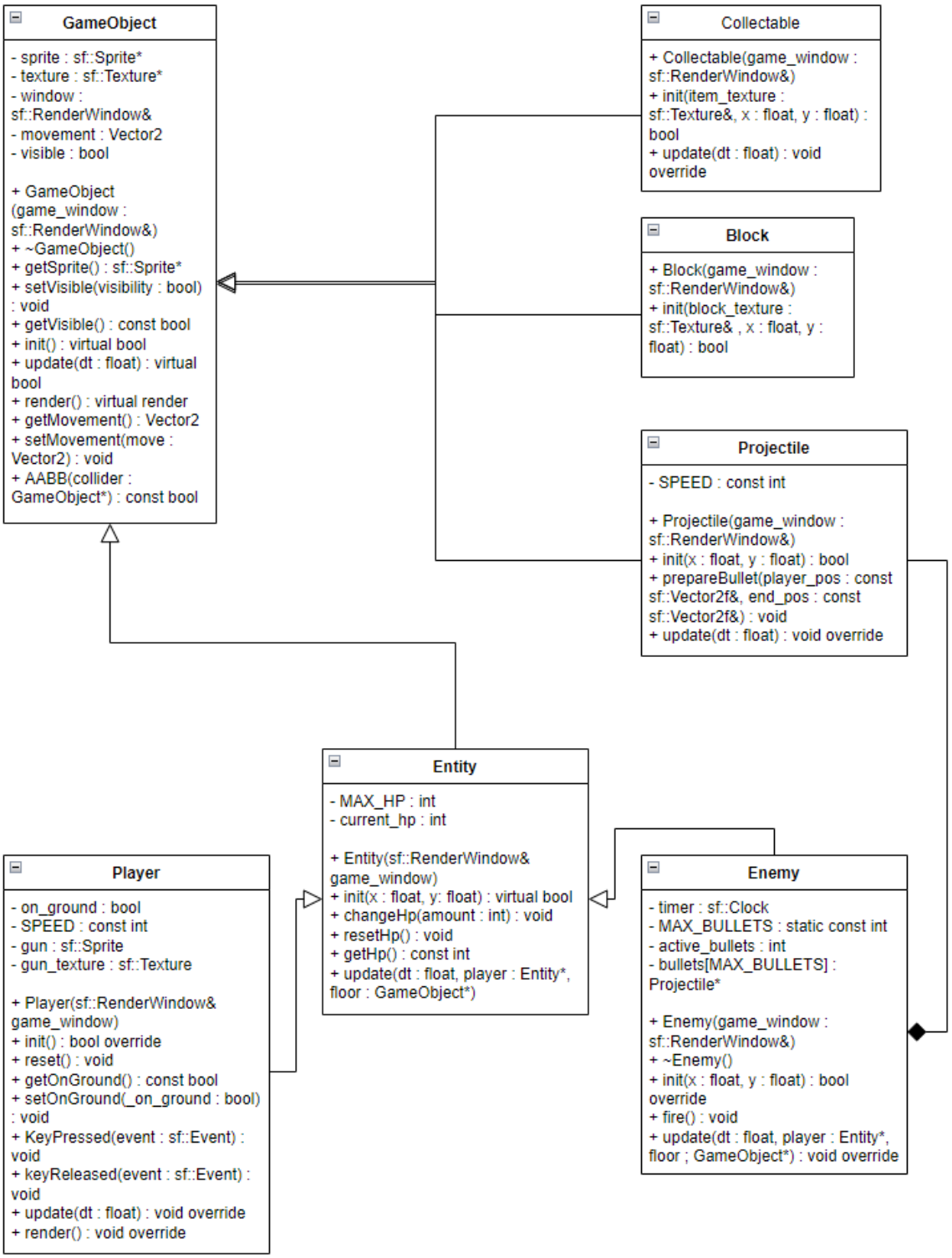
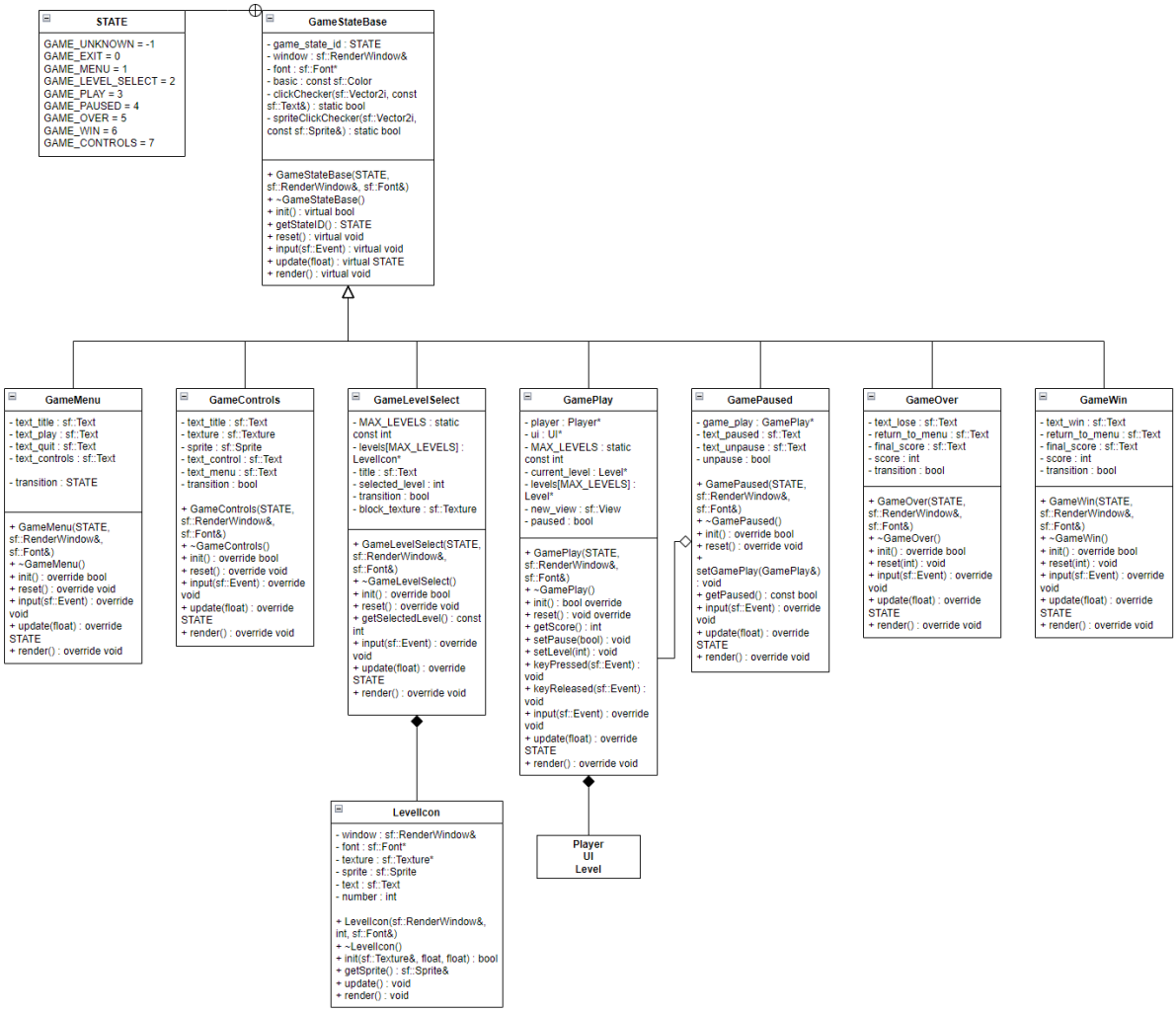
C++ Platformer Planning

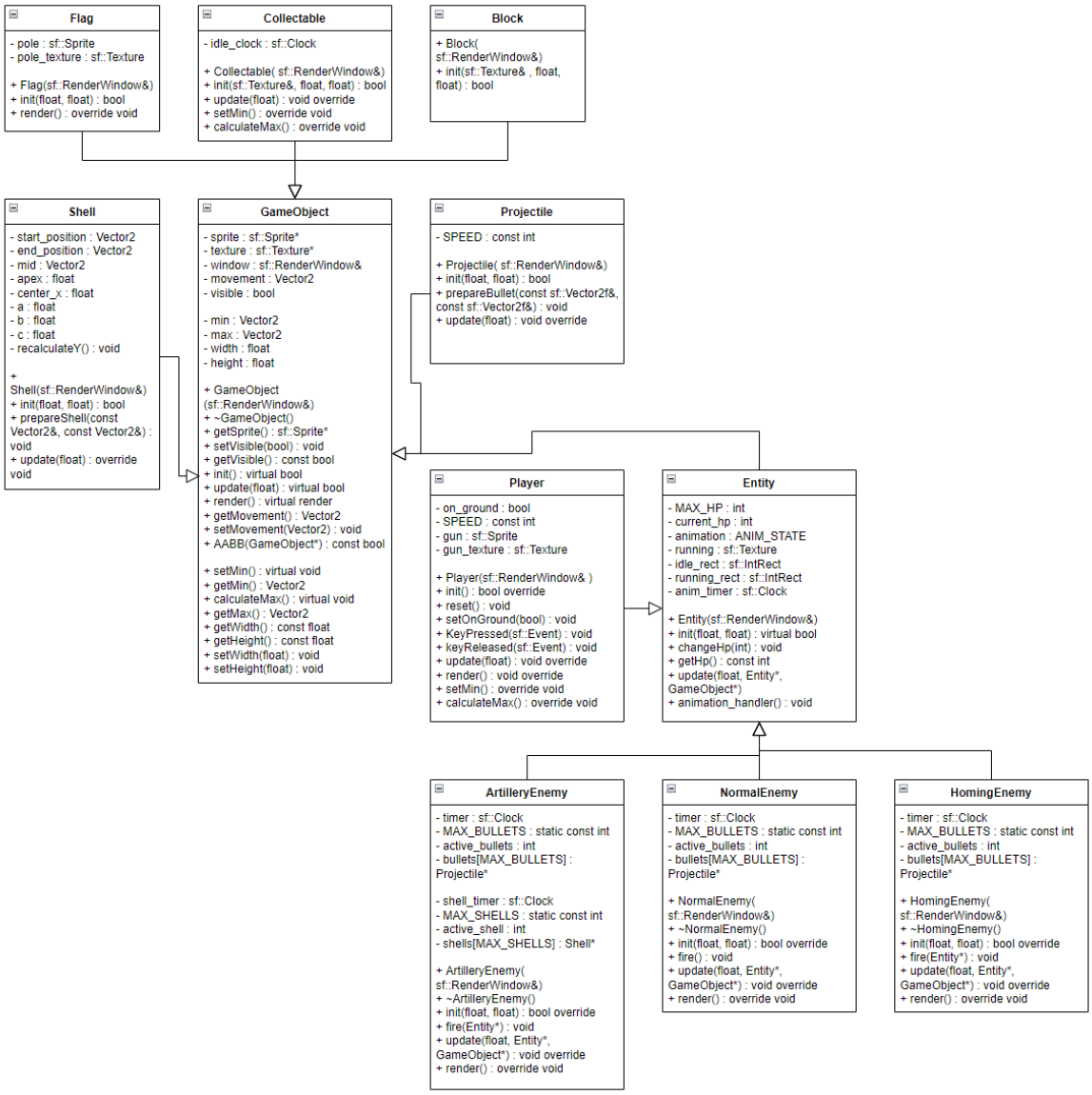
UML Before:




GameObject
- sprite : sf::Sprite*
- texture : sf::Texture*
- window : sf::RenderWindow&
- movement : Vector2
- visible : bool
+ GameObject (game_window : sf::RenderWindow&)
+ ~GameObject()
+ getSprite() : sf::Sprite*
+ setVisible(visibility : bool) : void
+ getVisible() : const bool
+ init() : virtual bool
+ update(dt : float) : virtual bool
+ render() : virtual render
+ getMovement() : Vector2
+ setMovement(move : Vector2) : void
+ AABB(collider : GameObject*) : const bool
Collectable
+ Collectable(game_window : sf::RenderWindow&)
+ init(item_texture : sf::Texture&, x : float, y : float) : bool
+ update(dt : float) : void override
Block
+ Block(game_window : sf::RenderWindow&)
+ init(block_texture : sf::Texture& , x : float, y : float) : bool
Projectile
- SPEED : const int
+ Projectile(game_window : sf::RenderWindow&)
+ init(x : float, y : float) : bool
+ prepareBullet(player_pos : const sf::Vector2f&, end_pos : const sf::Vector2f&) : void
+ update(dt : float) : void override
Entity
- MAX_HP : int
- current_hp : int
+ Entity(sf::RenderWindow& game_window)
+ init(x : float, y: float) : virtual bool
+ changeHp(amount : int) : void
+ resetHp() : void
+ getHp() : const int
+ update(dt : float, player : Entity*, floor : GameObject*)
Player
- on_ground : bool
- SPEED : const int
- gun : sf::Sprite
- gun_texture : sf::Texture
+ Player(sf::RenderWindow& game_window)
+ init() : bool override
+ reset() : void
+ getOnGround() : const bool
+ setOnGround(_on_ground : bool) : void
+ KeyPressed(event : sf::Event) : void
+ keyReleased(event : sf::Event) : void
+ update(dt : float) : void override
+ render() : void override
Enemy
- timer : sf::Clock
- MAX_BULLETS : static const int
- active_bullets : int
- bullets[MAX_BULLETS] : Projectile*
+ Enemy(game_window : sf::RenderWindow&)
+ ~Enemy()
+ init(x : float, y : float) : bool override
+ fire() : void
+ update(dt : float, player : Entity*, floor ; GameObject*) : void override

UML After:


STATE
GAME_UNKNOWN = -1
GAME_EXIT = 0
GAME_MENU = 1
GAME_LEVEL_SELECT = 2
GAME_PLAY = 3
GAME_PAUSED = 4
GAME_OVER = 5
GAME_WIN = 6
GAME_CONTROLS = 7
GameStateBase
- game_state_id : STATE
- window : sf::RenderWindow&
- font : sf::Font*
- basic : const sf::Color
- clickChecker(sf::Vector2i, const sf::Text&) : static bool
- spriteClickChecker(sf::Vector2i, const sf::Sprite&) : static bool
+ GameStateBase(STATE, sf::RenderWindow&, sf::Font&)
+ ~GameStateBase()
+ init() : virtual bool
+ getStateID() : STATE
+ reset() : virtual void
+ input(sf::Event) : virtual void
+ update(float) : virtual STATE
+ render() : virtual void
GameMenu
- text_title : sf::Text
- text_play : sf::Text
- text_quit : sf::Text
- text_controls : sf::Text
- transition : STATE
+ GameMenu(STATE, sf::RenderWindow&, sf::Font&)
+ ~GameMenu()
+ init() : override bool
+ reset() : override void
+ input(sf::Event) : override void
+ update(float) : override STATE
+ render() : override void
GameControls
- text_title : sf::Text
- texture : sf::Texture
- sprite : sf::Sprite
- text_control : sf::Text
- text_menu : sf::Text
- transition : bool
+ GameControls(STATE, sf::RenderWindow&, sf::Font&)
+ ~GameControls()
+ init() : override bool
+ reset() : override void
+ input(sf::Event) : override void
+ update(float) : override STATE
+ render() : override void
GameLevelSelect
- MAX_LEVELS : static const int
- levels[MAX_LEVELS] : LevelIcon*
- title : sf::Text
- selected_level : int
- transition : bool
- block_texture : sf::Texture
+ GameLevelSelect(STATE, sf::RenderWindow&, sf::Font&)
+ ~GameLevelSelect()
+ init() : override bool
+ reset() : override void
+ getSelectedLevel() : const int
+ input(sf::Event) : override void
+ update(float) : override STATE
+ render() : override void
LevelIcon
- window : sf::RenderWindow&
- font : sf::Font*
- texture : sf::Texture*
- sprite : sf::Sprite
- text : sf::Text
- number : int
+ LevelIcon(sf::RenderWindow&, int, sf::Font&)
+ ~LevelIcon()
+ init(sf::Texture&, float, float) : bool
+ getSprite() : sf::Sprite&
+ update() : void
+ render() : void
GamePlay
- player : Player*
- ui : UI*
- MAX_LEVELS : static const int
- current_level : Level*
- levels[MAX_LEVELS] : Level*
- new_view : sf::View
- paused : bool
+ GamePlay(STATE, sf::RenderWindow&, sf::Font&)
+ ~GamePlay()
+ init() : bool override
+ reset() : void override
+ getScore() : int
+ setPause(bool) : void
+ setLevel(int) : void
+ keyPressed(sf::Event) : void
+ keyReleased(sf::Event) : void
+ input(sf::Event) : override void
+ update(float) : override STATE
+ render() : override void
Player
UI
Level
GamePaused
- game_play : GamePlay*
- text_paused : sf::Text
- text_unpause : sf::Text
- unpause : bool
+ GamePaused(STATE, sf::RenderWindow&, sf::Font&)
+ ~GamePaused()
+ init() : override bool
+ reset() : override void
+ setGamePlay(GamePlay&) : void
+ getPaused() : const bool
+ input(sf::Event) : override void
+ update(float) : override STATE
+ render() : override void
GameOver
- text_lose : sf::Text
- return_to_menu : sf::Text
- final_score : sf::Text
- score : int
- transition : bool
+ GameOver(STATE, sf::RenderWindow&, sf::Font&)
+ ~GameOver()
+ init() : override bool
+ reset(int) : void
+ input(sf::Event) : override void
+ update(float) : override STATE
+ render() : override void
GameWin
- text_win : sf::Text
- return_to_menu : sf::Text
- final_score : sf::Text
- score : int
- transition : bool
+ GameWin(STATE, sf::RenderWindow&, sf::Font&)
+ ~GameWin()
+ init() : override bool
+ reset(int) : void
+ input(sf::Event) : override void
+ update(float) : override STATE
+ render() : override void



Comparison:

GameStates:

The main difference between the GameState classes is the number. As I created multiple levels, I add the GameLevelSelect class so there was a screen to select a level from. I also added the ability to pause whilst in gameplay. The controls screen was added as I remembered about the “Have an introductory screen with instructions” criteria.

GameObjects:

The main difference between the GameObject class itself is the addition of several variables to help me with performing collision detections. This include the min, max, width and height variables specifically. There is also the addition of the Shell class which is used by the ArtilleryEnemy exclusively. This class creates a parabola using 3 points in the Gameworld, which is then used to create the curved trajectory the Shell follows. As I created extra levels. I added a flag which became the new end goal, instead of collecting all the gems. 2 more enemy types were added to make the game more interesting, which are the Artillery (previously mentioned) and the Homing enemy, which fires bullets at the player.

Side note:

You will notice there are some Physics classes in my repo which I haven't documented here. I spent some time researching 2D physics engines and tried to implement one but was ultimately unsuccessful. I've kept the classes there more as a reference for myself for the future.

Pseudocode:

Platformer movement Pseudocode

```
gameplay()
        Sprite sprite
        Vector2 movement
        bool on_ground
        Gameobjects platform[]


keyPress(event)
        if event is D:
                movement.x = 5
        if event is A:
                movement.x = -5

        if event is SPACE and on_ground:
                movement.y = -5


keyRelease(event)
        if event is D:
                movement.x = 0;
        if event is A:
                movement.x = 0;


update(float dt)


        for platform in array:
                collision(sprite, platform)
```

```
        // Slowly adding to Y value creates a curved trajectory
        // If also moving on the X axis
        movement.y += 5 * dt

        if on_ground:
                movement.y = 0

        move sprite (movement.x * dt * SPEED, movement.y * dt * SPEED)

collision(sprite, platform)
        if sprite hits top of platform:
                sprite.y = platform.y + sprite.height
                sprite.on_ground = true
        if sprite hits left side of platform:
                sprite.x = platform.x - sprite.width
        if sprite hits right side of platform:
                sprite.x = platform.x + platform.width
        if sprite hits bottom of platform:
                sprite.y = 2
```