# C++ Platformer Planning

# Diagram Description automatically generatedUML Before:

# Diagram Description automatically generatedUML After: Diagram, table, schematic Description automatically generated

Diagram, schematic

Description automatically generated

## 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