

The background is a light gray gradient. It features several realistic water droplets of various sizes, some with highlights and shadows, scattered across the frame. In the upper center, there is a faint, circular, textured pattern that resembles a ripple or a lens flare.

Tower Defense

Mini Project 2 Package




Before we start,

Announcements

- You should have finished installing Allegro5 and set up your IDE on your own computer last semester in I2P course.
- If you did not take the course, see the [Tutorial](#) and videos.
- Our template requires **Allegro5** and **C++11** and you should compile and run the template successfully beforehand.
- If you use Visual Studio, you can download the project directly: [Visual Studio Project Template](#)




Outline

- Quick review
 - Resources
 - Scenes
 - Objects & Sprites
 - Objects & Controls
 - Template & Code structure
 - Goal & Grading Policy
- 



Outline

- Quick review
 - Resources
 - Scenes
 - Objects & Sprites
 - Objects & Controls
 - Template & Code structure
 - Goal & Grading Policy
- 

The background is a light gray gradient. It is decorated with numerous realistic water droplets of various sizes, some clustered in the top-left and bottom-right corners. A faint, circular, embossed-style logo is centered in the upper half of the image.

Quick Review

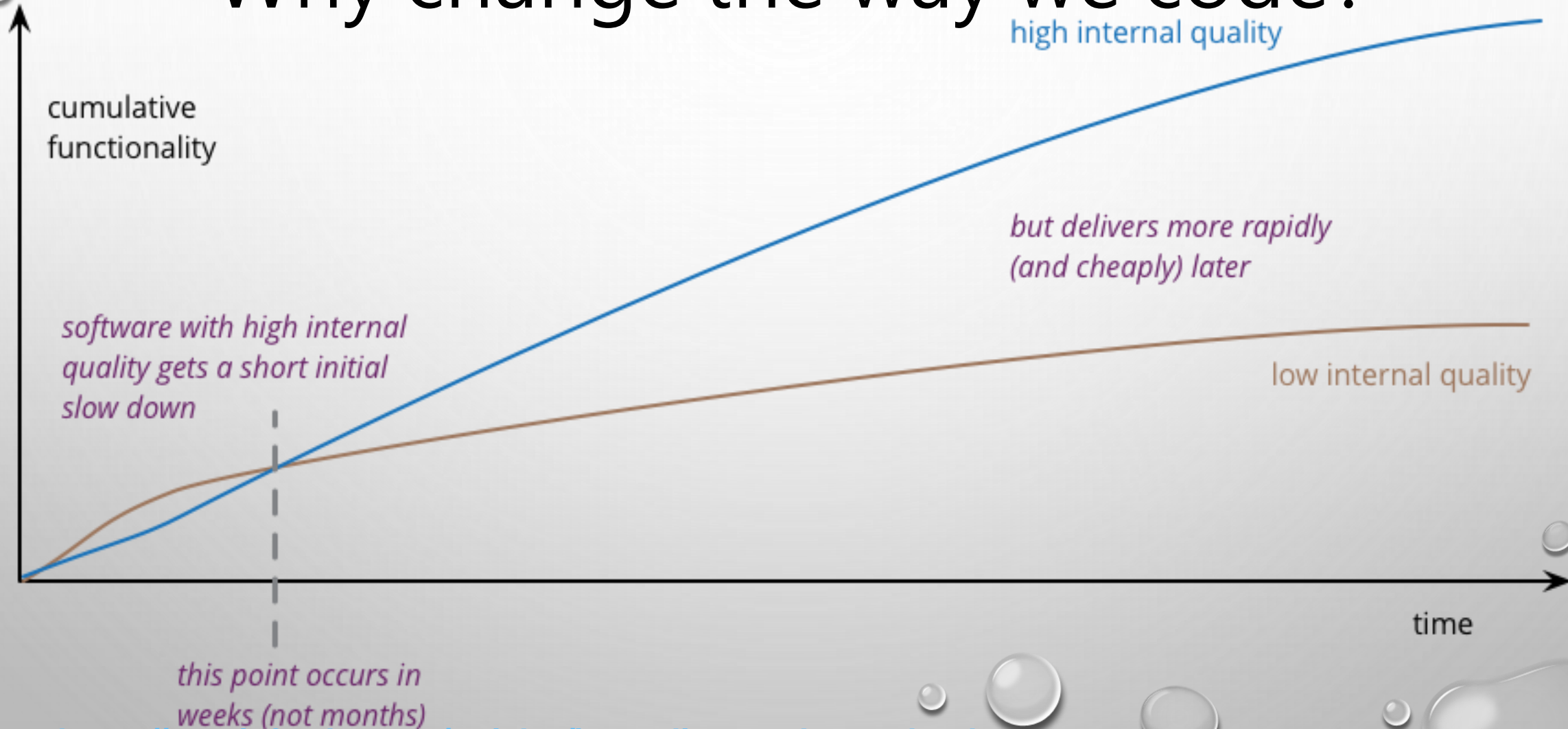
Allegro5 in C

Program Flow in Allegro5

- Your codes are still sequential.
- Initialize → loop (Wait for event → Process event → Draw) → Destroy



Why change the way we code?



The background is a light gray gradient. It features several realistic water droplets of various sizes, some with highlights and shadows, scattered across the frame. In the upper center, there is a faint, circular, textured pattern that resembles a ripple or a lens flare.

Quick Demo

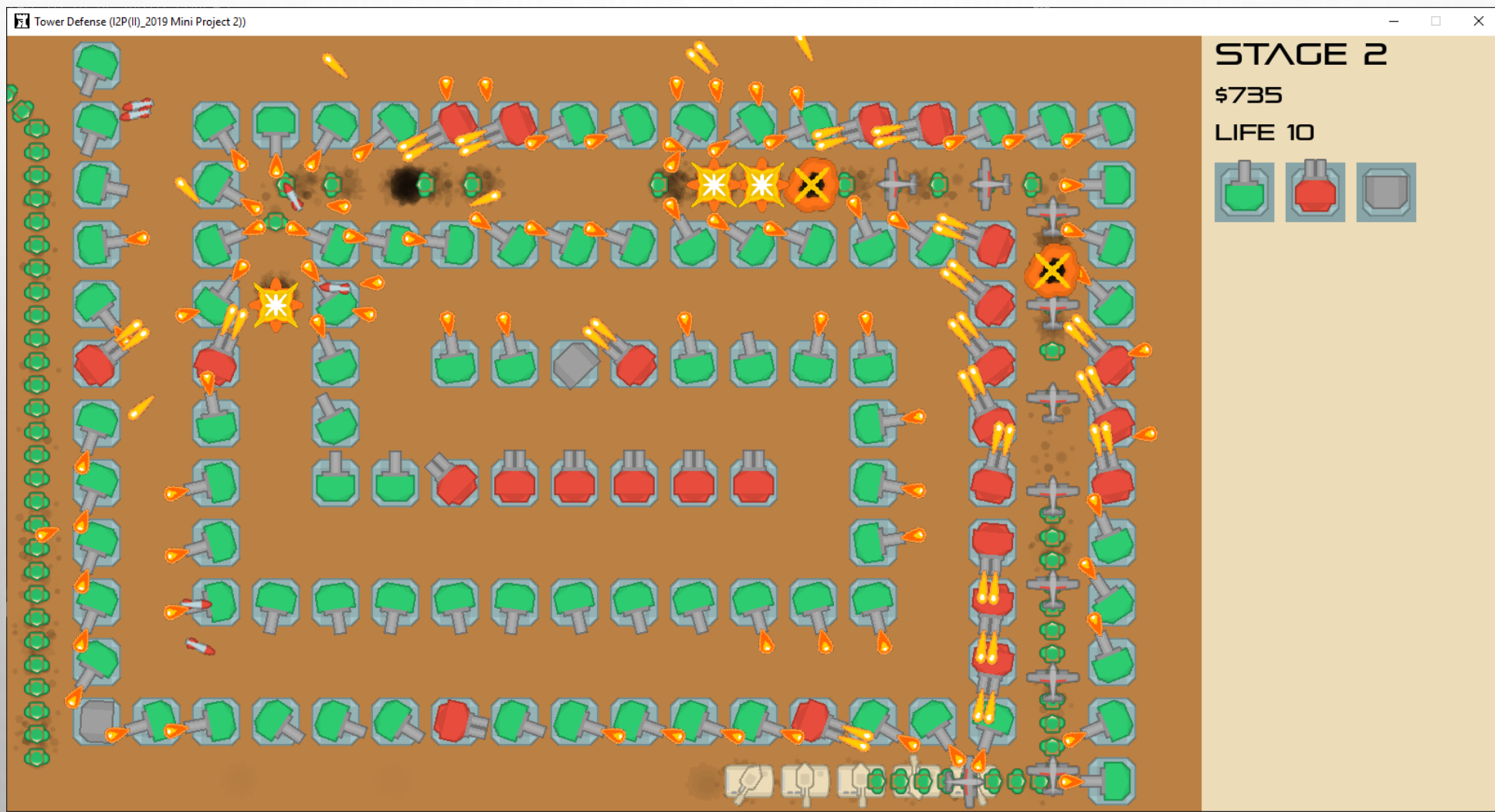
Tower Defense game demo

The background is a light gray gradient. It is decorated with numerous realistic water droplets of various sizes, some clustered in the top left and bottom right corners. A faint, large circular pattern, resembling a ripple or a stylized sun, is centered in the upper half of the image.

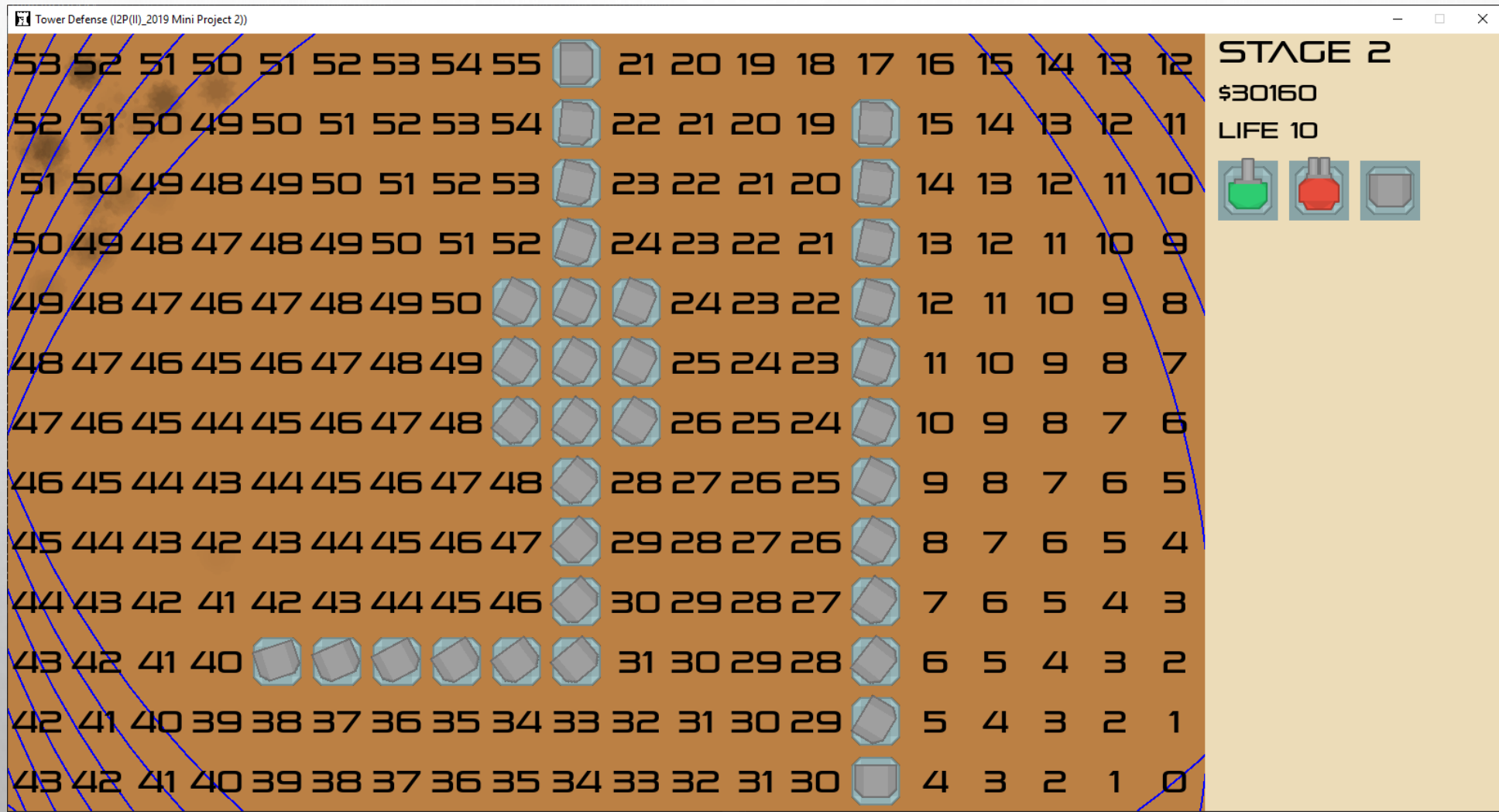
What do we care?

and what we don' t care?

Template Preview




Debug Mode





Outline

- Quick review
 - **Resources**
 - Scenes
 - Objects & Sprites
 - Objects & Controls
 - Template & Code structure
 - Goal & Grading Policy
- 

Resources

- Specify only what type of resources and where can we load them.



Images (Bitmap)



Audios (Samples)



Fonts

Resources Management

- Manually loading / destroying resources is unnecessary and causes memory leak if we are not careful enough.

```
ALLEGRO_BITMAP* img = al_load_bitmap("img.png");  
if (!img)  
    game_abort("failed to load image: img.png");  
//...  
al_destroy_bitmap(img);
```


Resources Management


- We can ignore resource management when using the wrapped **Resources** class: more convenient and less error prone.

➔

```
Resources::GetInstance().GetBitmap("img.png");  
//...  
// Automatically free resources.
```

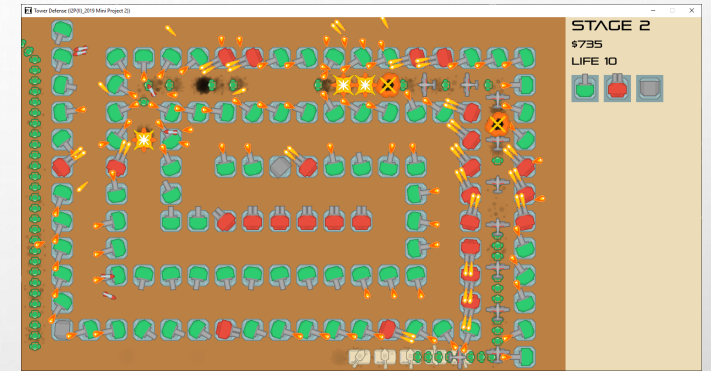


Outline

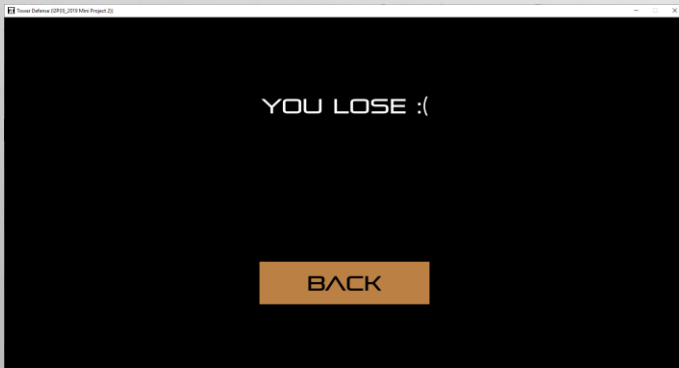
- Quick review
 - Resources
 - **Scenes**
 - Objects & Sprites
 - Objects & Controls
 - Template & Code structure
 - Goal & Grading Policy
- 

Scenes

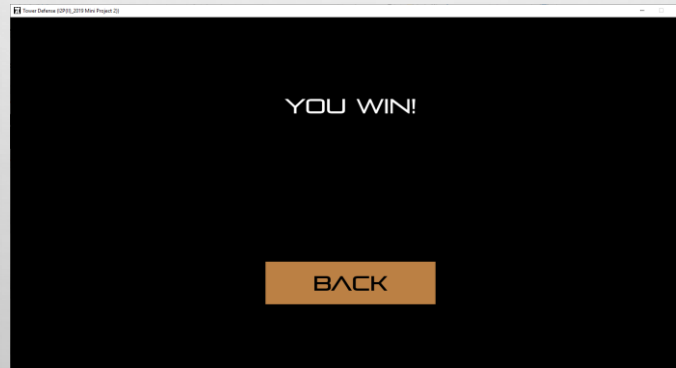
- All scenes should be independent.
- Change between scenes with only a function call.



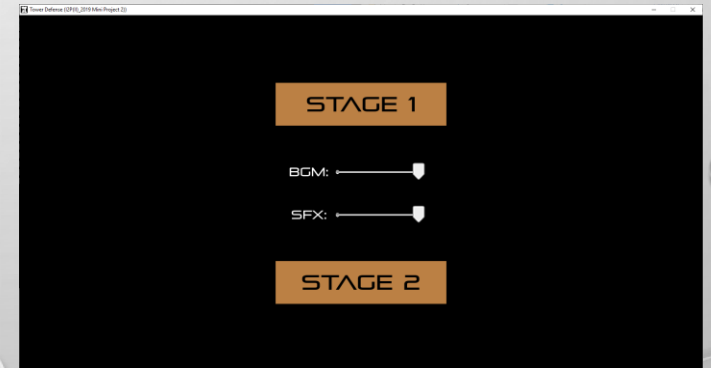
Play Scene



Lose Scene



Win Scene



Stage Select Scene

Multiple Scenes

- Manually checking which scene to update / draw is redundant, and we cannot have same variable names in different scenes.

```
void game_update(void) {  
    if (active_scene == SCENE_A) {  
        //...  
    } else if (active_scene == SCENE_B) {  
        //...  
    } // Maybe we have up to 5 scenes...  
}  
// The same structure above is also used in  
`game_draw`, `game_change_scene`, and various events
```

Multiple Scenes


- We can ignore the existence of other scenes and see each scene as independent **IScene** class: more encapsulation.



```
class SceneA final : public Engine::IScene {  
public:  
    explicit SceneA() = default;  
    void Initialize() override;  
    void Terminate() override;  
    void Update() override;  
    void Draw() const override;  
};
```

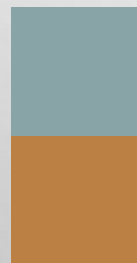


Outline

- Quick review
 - Resources
 - Scenes
 - **Objects & Sprites**
 - Objects & Controls
 - Template & Code structure
 - Goal & Grading Policy
- 

Controls & Objects

- Static images
- Images that can move, rotate, ...
- Buttons
- Label (Text)



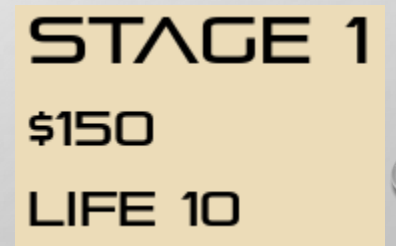
Image



Sprite



ImageButton



Label (Text)

Objects & Sprites

- A simple sprite requires too much code.

```
void draw_movable_object(MovableObject obj) {  
    if (obj.hidden) return;  
    al_draw_bitmap(obj.img, round(obj.x - obj.w / 2),  
        round(obj.y - obj.h / 2), 0);  
}  
void game_update() {  
    for (i = 0; i < MAX_OBJ; i++) {  
        if (objs[i].hidden) continue;  
        objs[i].x += objs[i].vx;  
        objs[i].y += objs[i].vy;  
    }  
}
```

Objects & Sprites

- We can define a class and specify some behaviors of the objects. Then, we can add and forget about it: one-liner for every object.



```
void SceneA::Shoot(int x, int y) {  
    AddNewObject(new Bullet(x, y));  
}
```



Outline

- Quick review
 - Resources
 - Scenes
 - Objects & Sprites
 - **Objects & Controls**
 - Template & Code structure
 - Goal & Grading Policy
- 

Objects & Controls

- A simple button requires too much code.

```
void on_mouse_down(int btn, int x, int y) {  
    if (btn == 1 && pnt_in_rect(x, y, btnX, btnY, btnW, btnH)) {  
        // Button clicked.  
    }  
}  
  
void game_draw() {  
    if (pnt_in_rect(mouse_x, mouse_y, btnX, btnY, btnW, btnH))  
        al_draw_bitmap(img_btn_in, btnX, btnY, btnW, btnH);  
    else  
        al_draw_bitmap(img_btn_out, btnX, btnY, btnW, btnH);  
}
```

Objects & Controls

- We can ignore the drawing and mouse-in detection. For buttons, we only want to know when it is clicked. Declaring a variable just for the button is also unnecessary: higher abstraction.



```
void SceneA::BtnOnClick() { // Button clicked. }  
void SceneA::Initialize() {  
    ImageButton* btn = new ImageButton("img_out.png", "img_in.png", 0, 0);  
    btn->SetOnClickCallback(std::bind(&SceneA::BtnOnClick, this));  
    AddNewControlObject(btn);  
}
```

Outline

- Quick review
- Resources
- Scenes
- Objects & Sprites
- Objects & Controls
- **Template & Code structure**
- Goal & Grading Policy

Template Naming Convention

- Usually, C++ uses snake case, but we use camel case here to distinguish between STL and self-defined code.
- `std::??? (snake_case)` → C++11 STL
- `al_???, ALLEGRO_???` → Allegro5 libraries' API.
- `Engine::??? (CamelCase)` → Our own defined wrapper
`::???` → Classes used in game.

Template Diagram

- [Class Diagram](#)
- [Engine Class Diagram](#)
- [Engine Class Diagram Minimized](#)
- [Game Class Diagram](#)
- [Game Class Diagram Minimized](#)
- [Game Class Diagram Minimized Annotated](#)

The background is a light gray gradient. It is decorated with numerous realistic water droplets of various sizes, some with highlights and shadows, scattered across the frame. In the upper center, there is a faint, circular logo or watermark that appears to contain a stylized 'E' or a similar symbol.

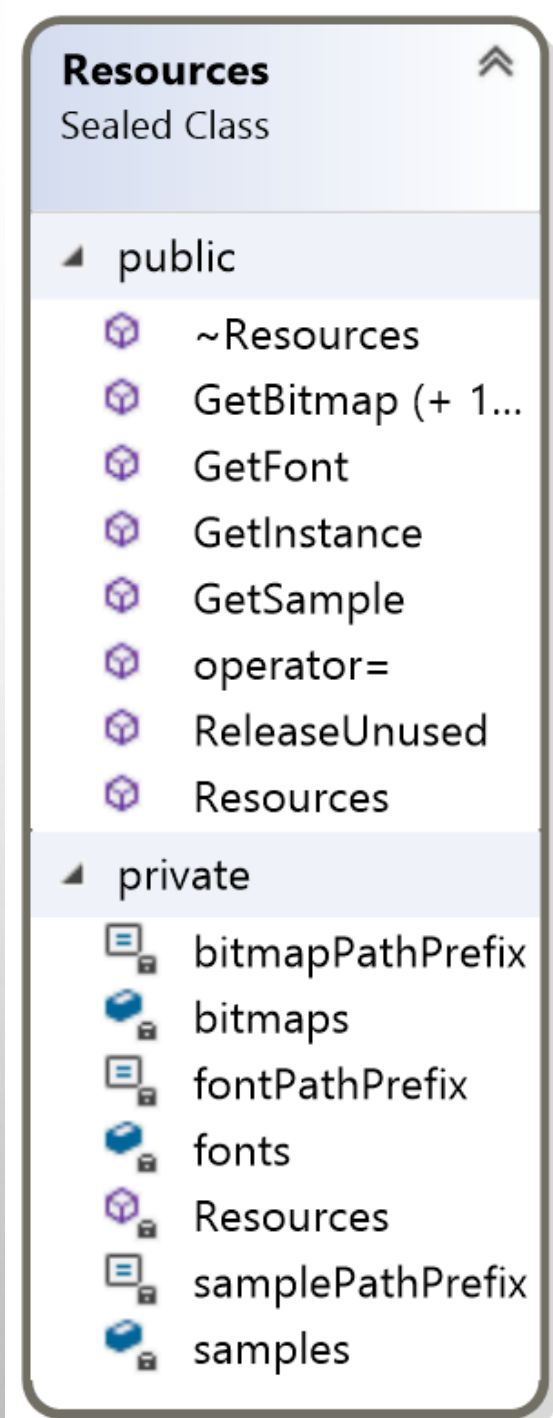
Engine code

Tower Defense

Template: Resources

Engine::Resources

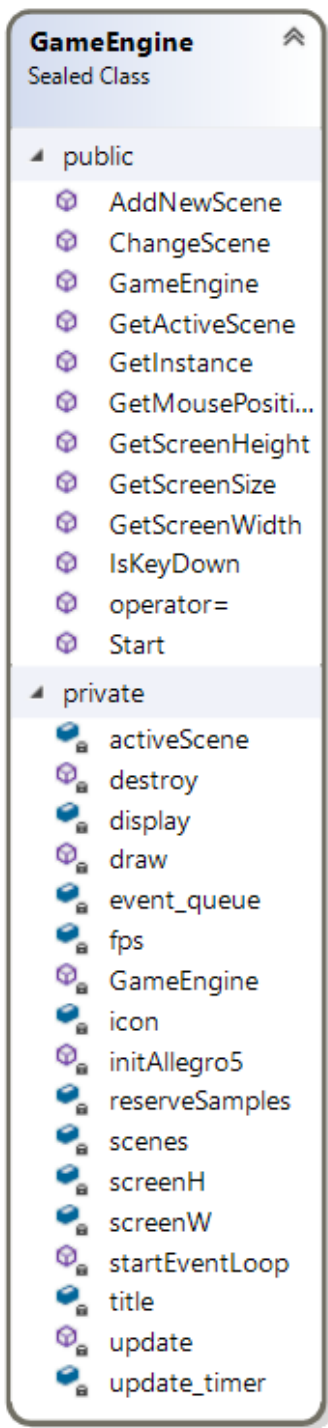
- Abstracts all resources loading and destroy.
- Resources can be retrieved from this class directly.



Template: Game Engine

Engine::GameEngine

- Abstracts the entire message loop
- Manages current scene and scene changes.



Template: IScene, Group

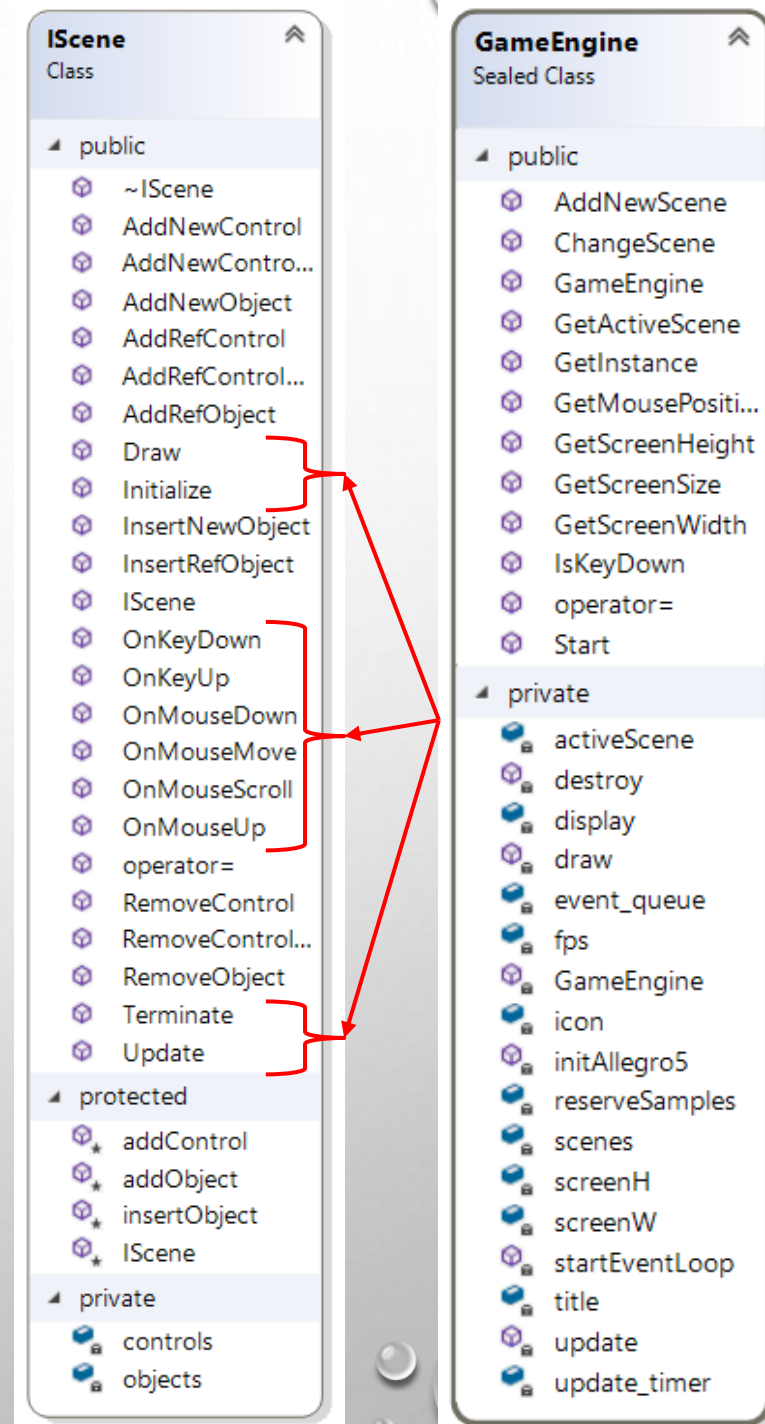
Engine::IScene

- Encapsulates a scene, must be inherited and customized.

Engine::Group

- Draw and update everything for you.

Note: We combined Group and IScene in this diagram



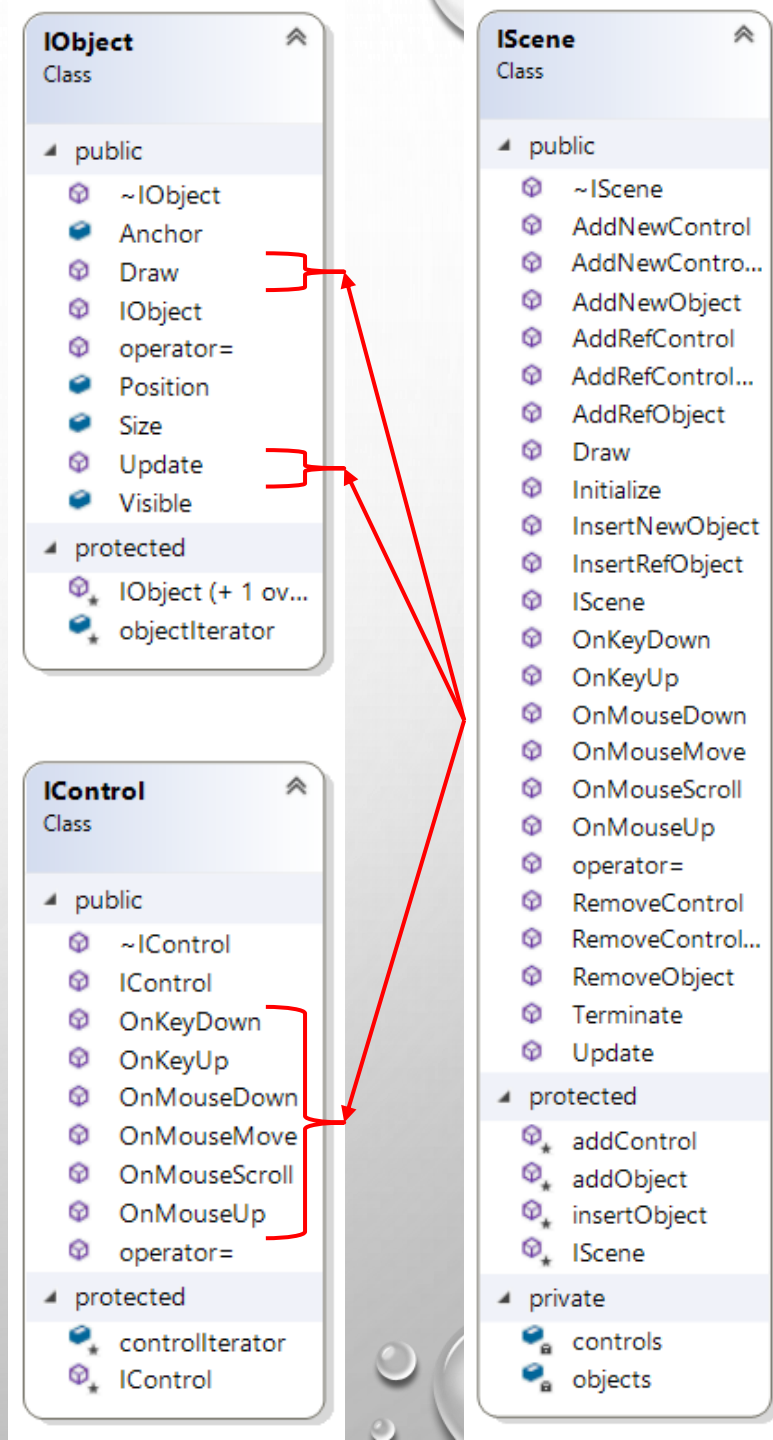
Template: IObject, IControl

Engine::IObject

- The base class of everything that can be drawn.

Engine::IControl

- The base class of everything that can receive events.



Template: Image, Sprite

Engine::Image :

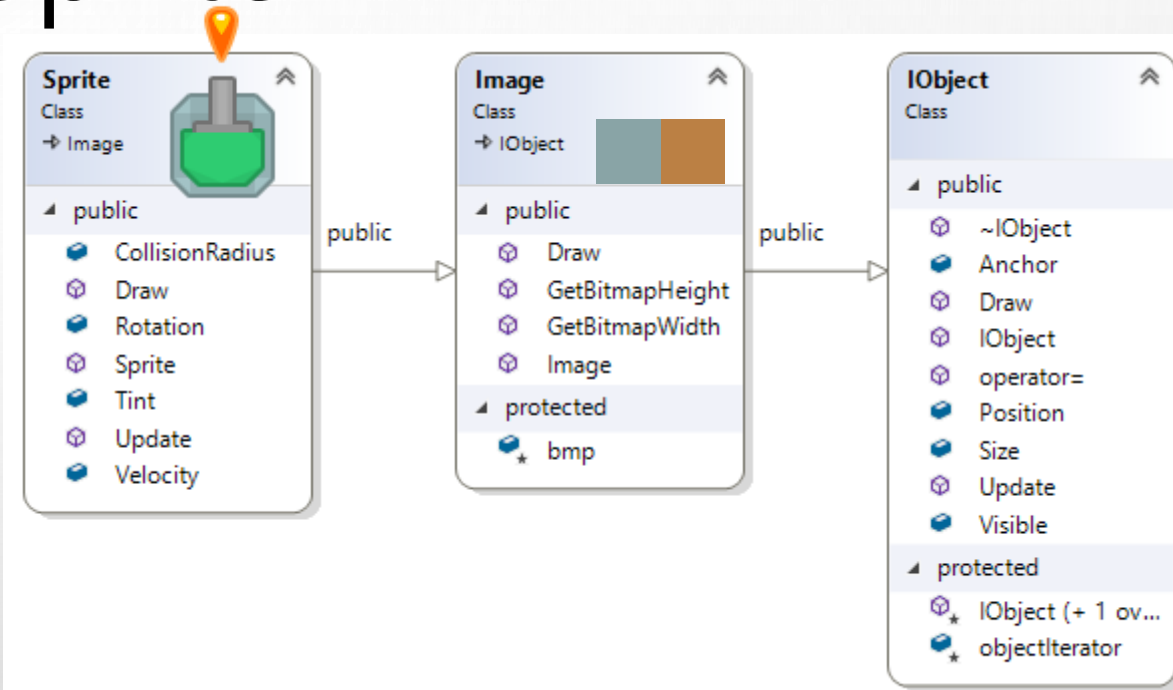
public Engine::IObject

- A simple static image object.

Engine::Sprite :

public Engine::Image

- Supports rotation, velocity, tint, and collision radius.



Template: Label, ImageButton

Engine::Label :

public Engine::IObject

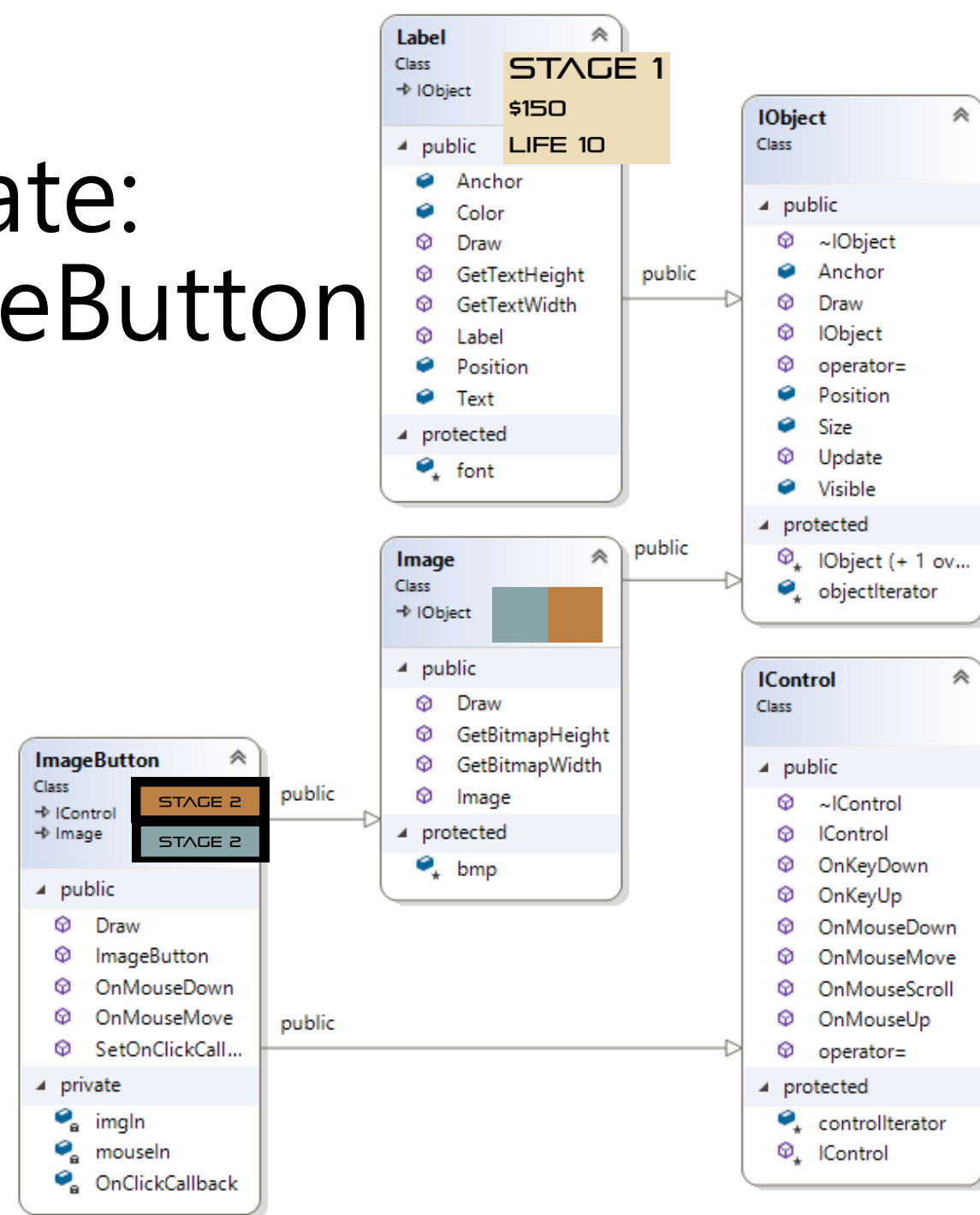
- A simple static text object.

Engine::ImageButton :

public Engine::IObject

public Engine::IControl

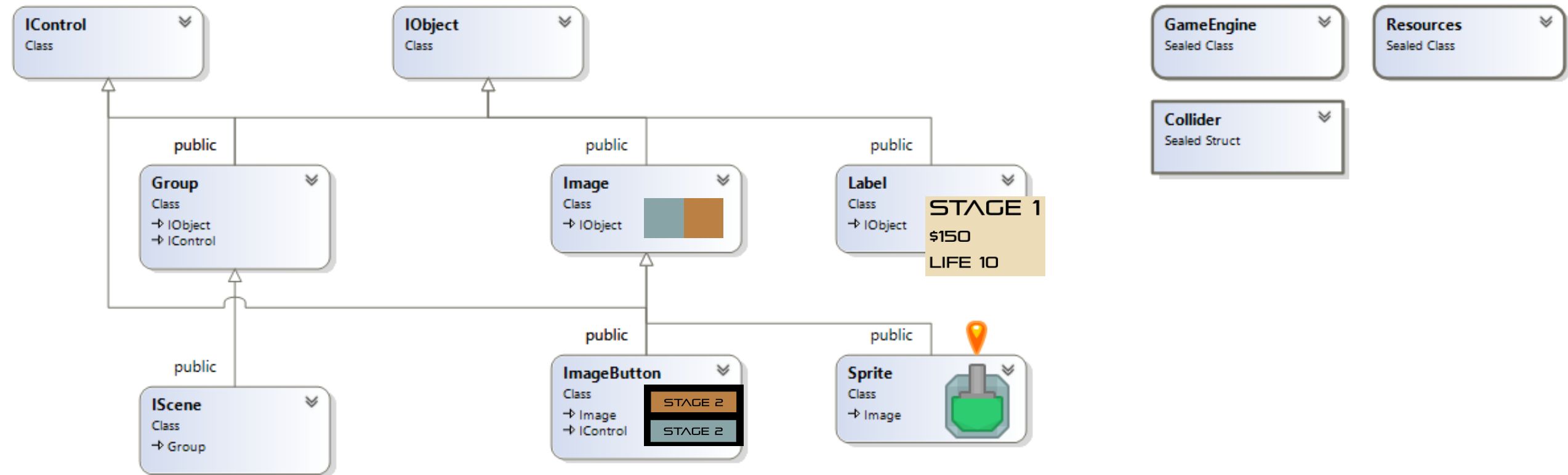
- A clickable button, changes image when mouse move.



Engine Diagram (Minimized)

(OnMouseMove,
OnMouseDown, ...)

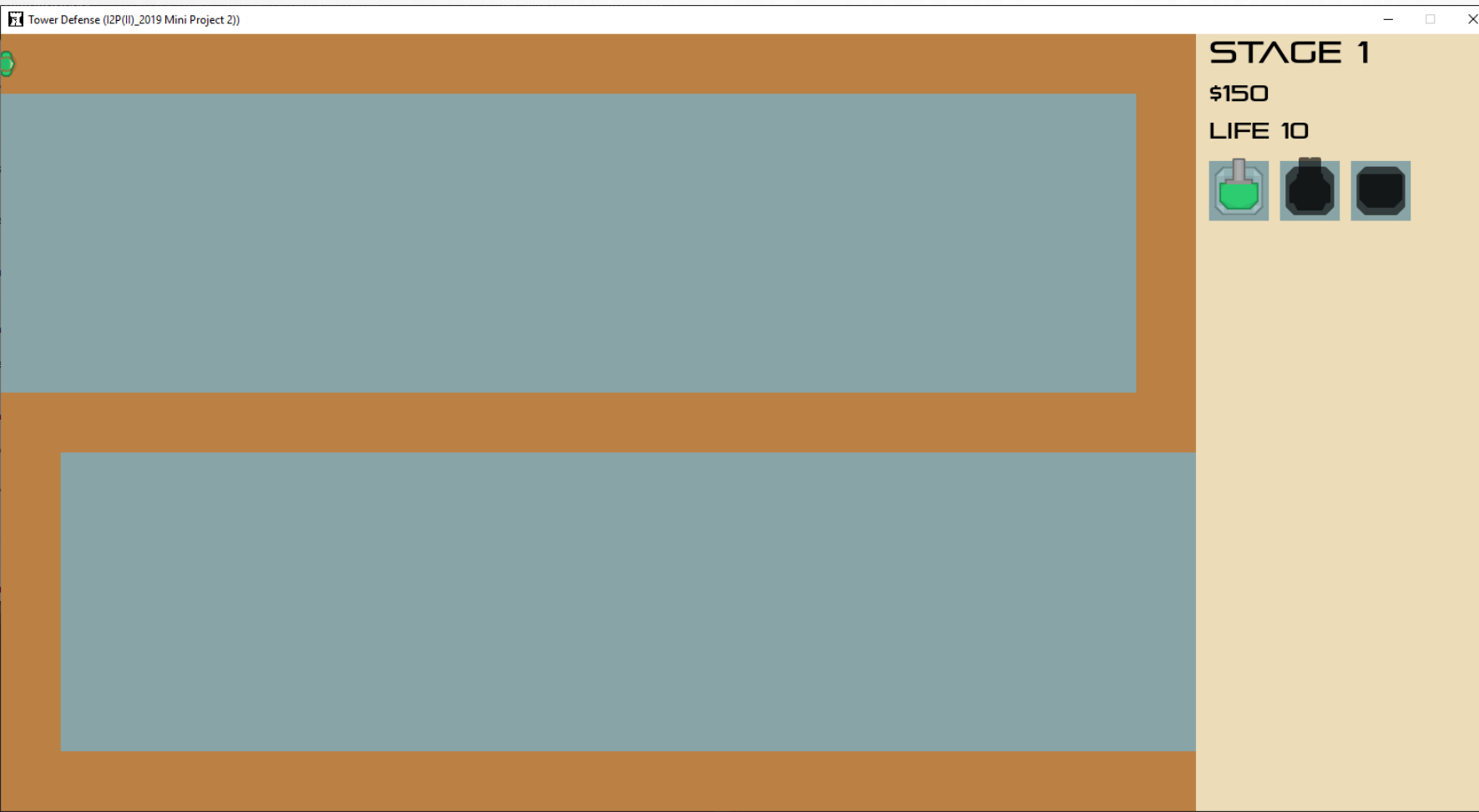
(Update, Draw)



The background is a light gray gradient. It features several realistic water droplets of various sizes, some with highlights and shadows, scattered across the frame. In the upper center, there is a faint, circular, textured pattern that resembles a ripple or a lens flare.

Game code

Tower Defense

[illegible]

```

00000000000000000000000000
11111111111111111111111110
11111111111111111111111110
11111111111111111111111110
11111111111111111111111110
11111111111111111111111110
00000000000000000000000000
01111111111111111111111111
01111111111111111111111111
01111111111111111111111111
01111111111111111111111111
01111111111111111111111111
00000000000000000000000000

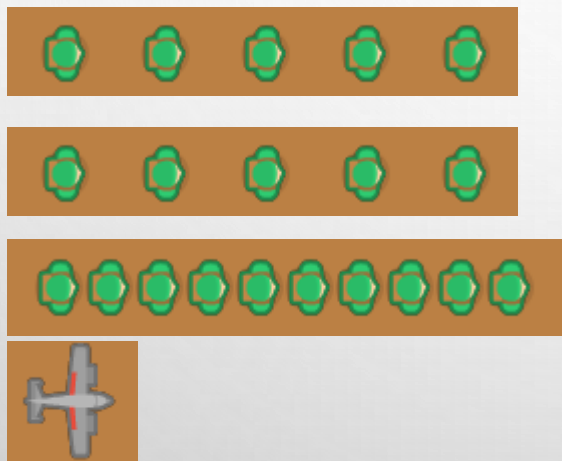
```

resources/map1.txt

Enemy file format

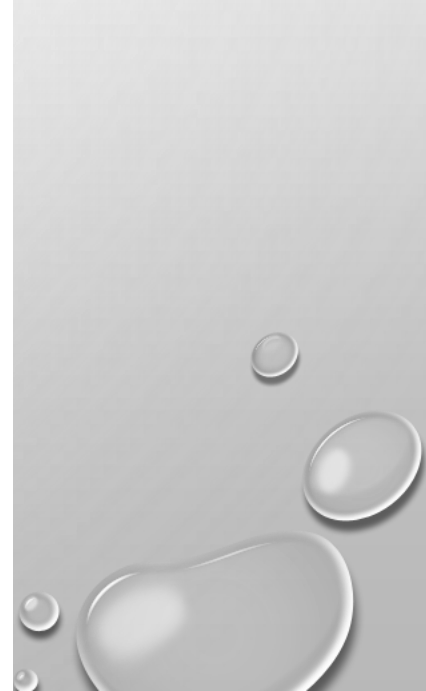
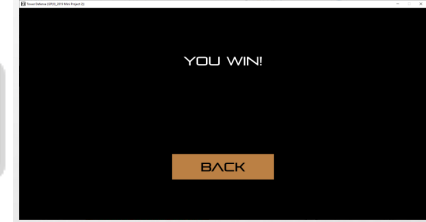
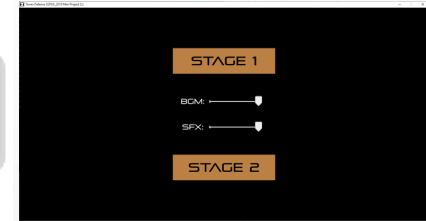
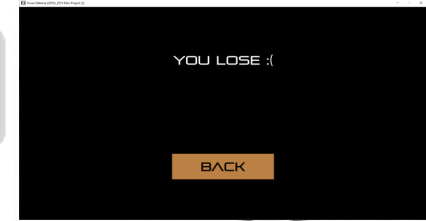
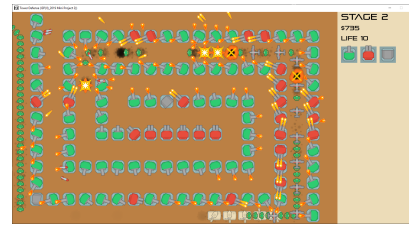
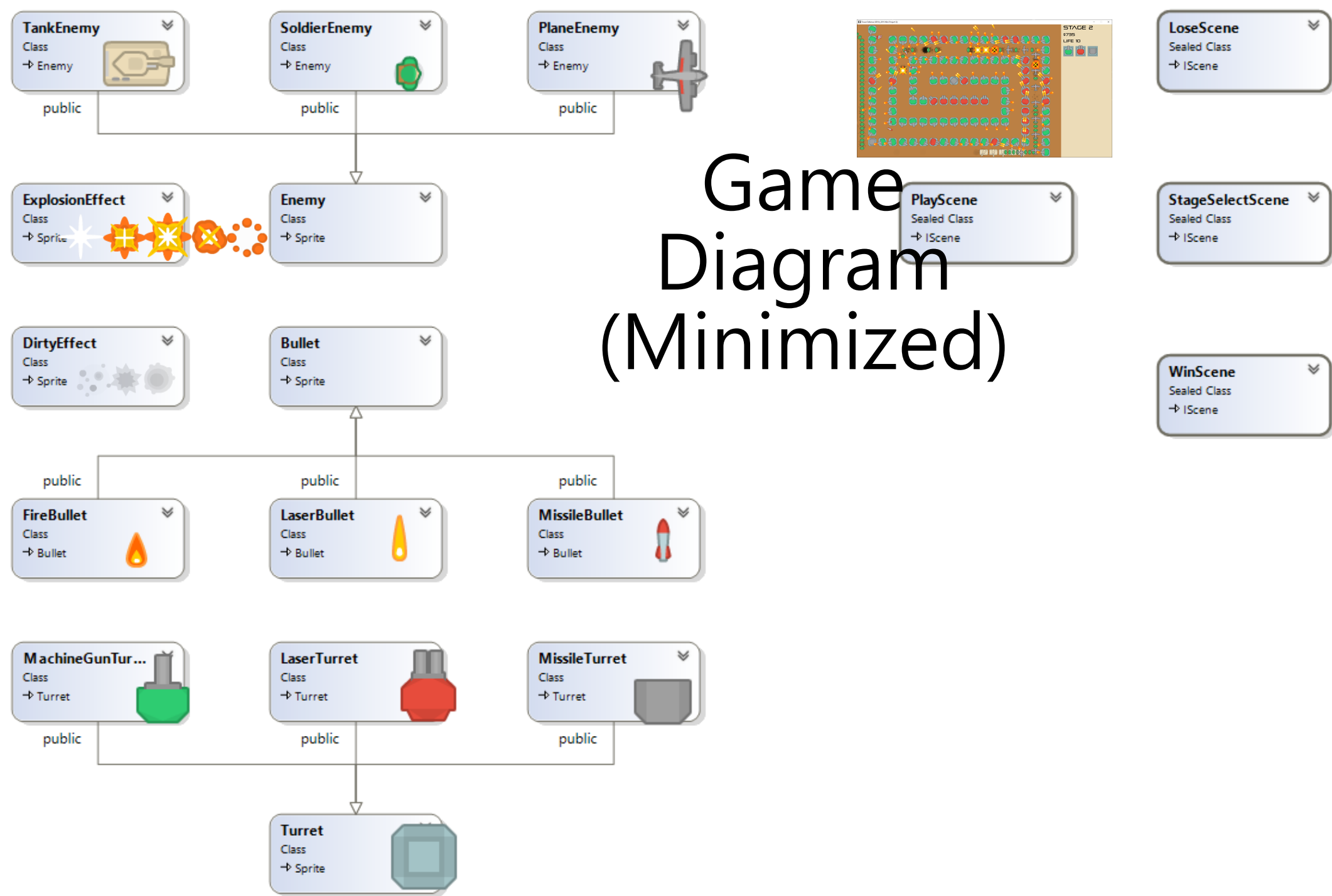
- EnemyType TimeDelayBetween Count

```
1 1 5  
0 2 1  
1 1 5  
0 2 1  
1 0.5 10  
0 6 1  
2 1 1  
0 2 1  
1 0.5 20  
0 12 1  
2 1 5  
0 2 1  
1 0.5 20
```



You should edit this file
after adding new enemy
resources/enemy1.txt

Game Diagram (Minimized)




Future of game programming

- Component system
 - Physics engine
 - Functional programming
 - Entity component system (ECS)
-
- However, OOP is still a concept that cannot be abandoned in most programs.



Outline

- Quick review
 - Resources
 - Scenes
 - Objects & Sprites
 - Objects & Controls
 - Template & Code structure
 - **Goal & Grading Policy**
- 

The background is a light gray gradient. It is decorated with numerous realistic water droplets of various sizes, some clustered in the top-left and bottom-right corners. A faint, circular logo is centered in the upper half of the image, featuring a stylized 'E' and the word 'PARK' below it.

Q&A

Feel free to ask any question.