Game Objects

Game State

Game Mode

Game Objects

(entities)

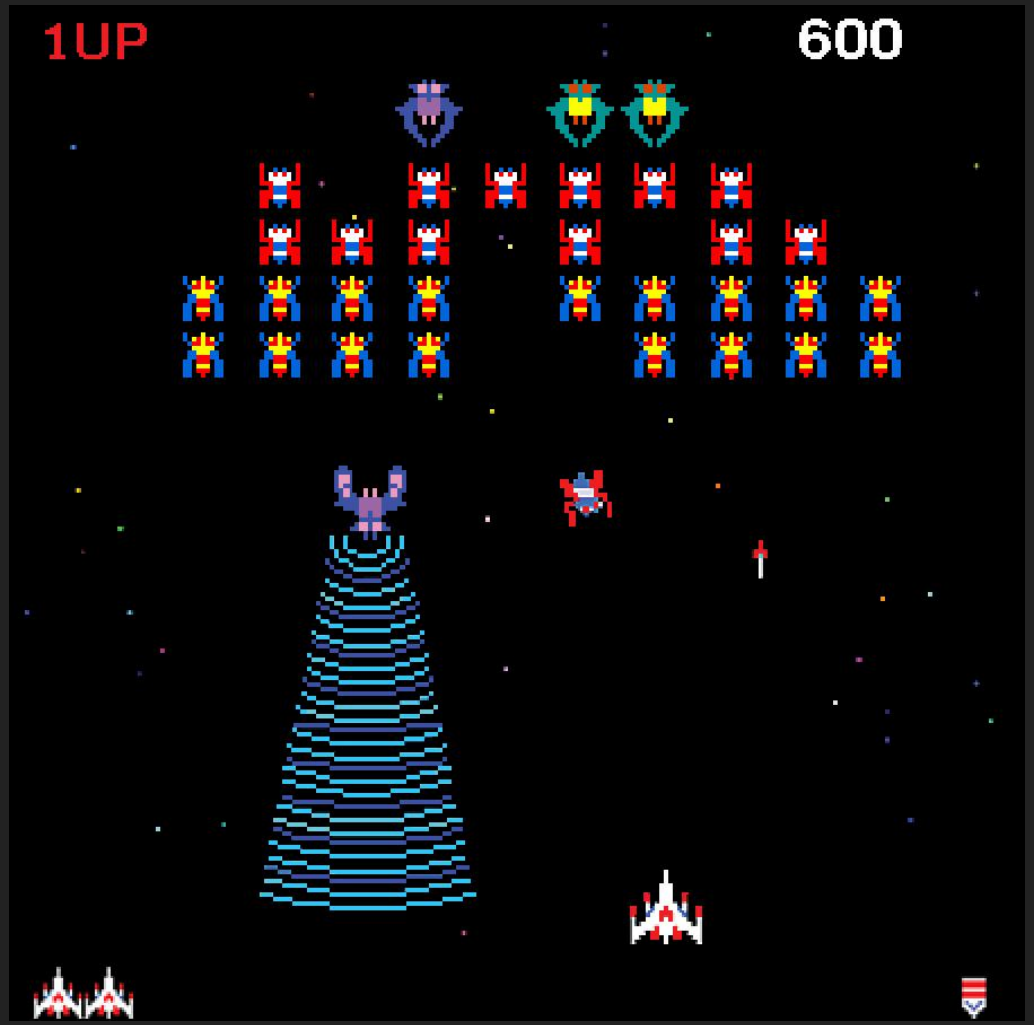
Entities help us to organize and manage game objects.

```
class Entity {
public:
    glm::vec3 position;
    glm::vec3 movement;
    float speed;

    GLuint textureID;

    Entity();

    void Update(float deltaTime);
    void Render(ShaderProgram *program);
};
```


1UP
600


250
100
20/20
Norad II
General Duke
Kills: 0
700/700
250/250
0
0
0
MENU

Object Pool

(optimizing for tons of objects)

Object Pool

Create/Allocate objects ahead of time.

Use a bool for active or not.

Maximum number of objects.

You can test with max amount of objects.

Less prone to memory leaks.

Object Pool

```
#define MAX_BULLETS 100;

int nextBullet = 0;
Entity bullets[MAX_BULLETS];

void initialize() {
    for (int i = 0; i < MAX_BULLETS; i++) {
        bullets[i].active = false;
    }
}

void fire() {
    bullet[nextBullet].position = // somewhere
    bullet[nextBullet].active = true;

    nextBullet++;
    if (nextBullet == MAX_BULLETS) nextBullet = 0;
}
```

Game State

entities, score, lives left, time left, etc.

Game State

```
struct GameState {
    Entity player;
    Entity enemies[10];
    Entity items[5];
    int score;
};

GameState state;
```

Game Mode

Game Mode

Arcade games typically feature an “attract” mode.
Also called “demo” mode. Sometimes seen in NES games.

Game plays itself, shows high scores, cut scenes, etc.

Rygar:
https://www.youtube.com/watch?v=jV2iT9LnCD4

Street Fighter II:
https://www.youtube.com/watch?v=TU1C1ihW2mg

Game Mode

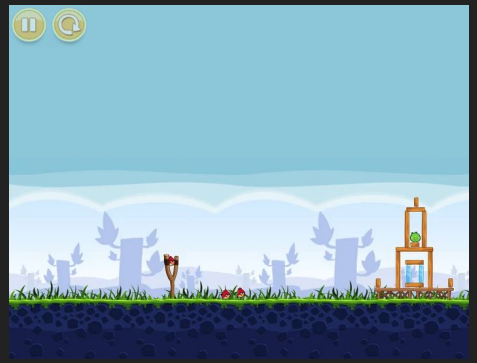
Modern Console/PC Games typically do not have Attract/Demo modes.

Main Menu

Chapter Select

Level Select

Cut Scene



Game Level

Win Screen

Game Mode

```
enum GameMode { MAIN_MENU, GAME_LEVEL, GAME_OVER };
GameMode mode = MAIN_MENU;
```

Game Mode

```
void ProcessInput() {
    switch (mode) {
        case MAIN_MENU:
            ProcessInputMainMenu();
            break;

        case GAME_LEVEL:
            ProcessInputGameLevel();
            break;

        case GAME_OVER:
            ProcessInputGameOver();
            break;
    }
}
```

Game Mode

```
void Update() {
    float ticks = (float)SDL_GetTicks() / 1000.0f;
    float deltaTime = ticks - lastTicks;
    lastTicks = ticks;

    switch (mode) {
        case MAIN_MENU:
            UpdateMainMenu(deltaTime);
            break;

        case GAME_LEVEL:
            UpdateGameLevel(deltaTime);
            break;

        case GAME_OVER:
            UpdateGameOver(deltaTime);
            break;
    }
}
```

Game Mode

```
void Render() {
    glClear(GL_COLOR_BUFFER_BIT);

   switch (mode) {
       case MAIN_MENU:
           RenderMainMenu();
           break;

       case GAME_LEVEL:
           RenderGameLevel();
           break;

       case GAME_OVER:
           RenderGameOver();
           break;
   }

    SDL_GL_SwapWindow(displayWindow);
}
```

Game Mode

(or make a class for each mode)

```
MainMenu mainMenu;
GameLevel gameLevel;
GameOver gameOver;

void Render() {
     switch (mode) {
          case GAME_LEVEL:
               gameLevel.Render();
          break;

          // .. other modes
     }
}
```

```
class GameLevel {
     GameState state;

     public void Render() {
          state.player.Render();
          for (int i = 0; i < enemies.length; i++) {
               state.enemies[i].Render();
          }
     }

     // Other stuff

}
```

Let’s Talk About

Game Mechanics

In-Class Exercise

Uno

(With New Mechanics)