

Graphics Foundations



Part 3.5

Non-uniform sprite sheets.

Uniform





Non-uniform

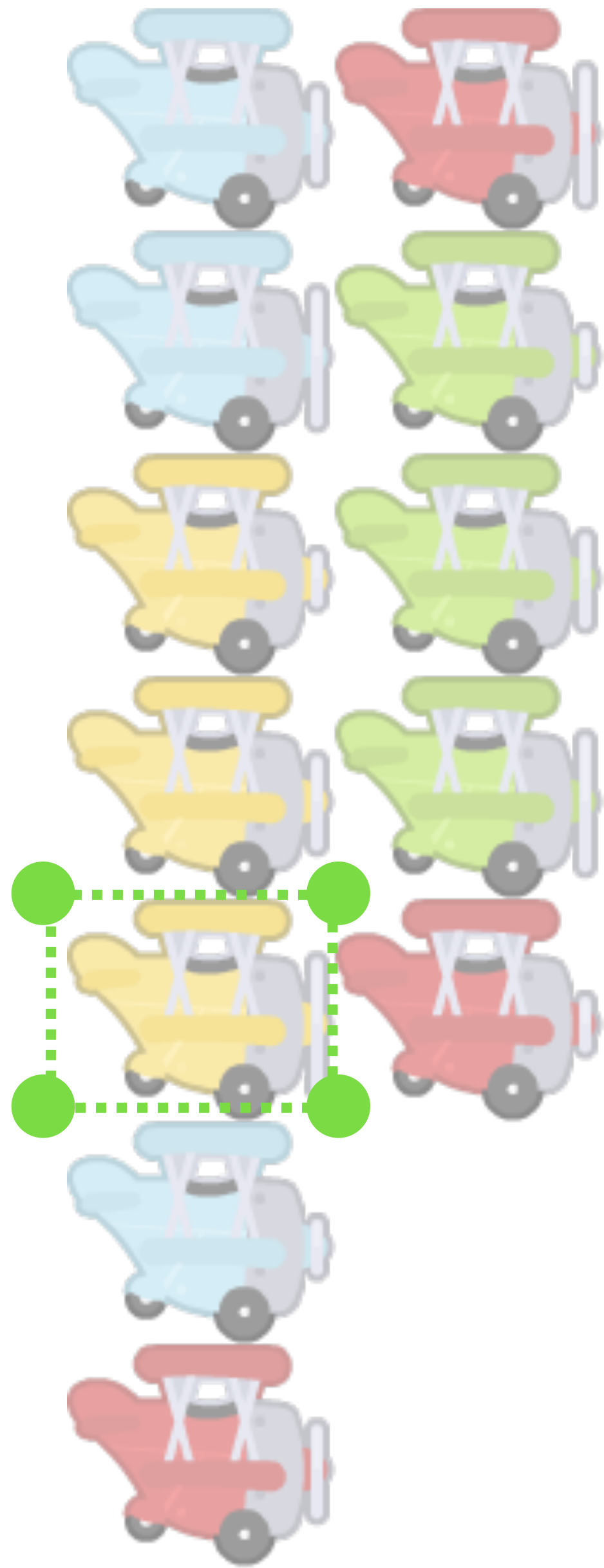


Need to keep a list of
coordinates for non-
uniform sprite sheets.

Texture atlas XML

```
<TextureAtlas imagePath="sheet.png">
  <SubTexture name="beam0.png" x="143" y="377" width="43" height="31"/>
  <SubTexture name="beam1.png" x="327" y="644" width="40" height="20"/>
  <SubTexture name="beam2.png" x="262" y="907" width="38" height="31"/>
  <SubTexture name="beam3.png" x="396" y="384" width="29" height="29"/>
  <SubTexture name="beam4.png" x="177" y="496" width="41" height="17"/>
  <SubTexture name="beam5.png" x="186" y="377" width="40" height="25"/>
  <SubTexture name="beam6.png" x="120" y="688" width="43" height="23"/>
  <SubTexture name="beamLong1.png" x="828" y="943" width="15" height="67"/>
  <SubTexture name="beamLong2.png" x="307" y="309" width="25" height="64"/>
  <SubTexture name="bold_silver.png" x="810" y="837" width="19" height="30"/>
  <SubTexture name="bolt_bronze.png" x="810" y="467" width="19" height="30"/>
  <SubTexture name="bolt_gold.png" x="809" y="437" width="19" height="30"/>
  <SubTexture name="buttonBlue.png" x="0" y="78" width="222" height="39"/>
  <SubTexture name="buttonGreen.png" x="0" y="117" width="222" height="39"/>
  <SubTexture name="buttonRed.png" x="0" y="0" width="222" height="39"/>
  <SubTexture name="buttonYellow.png" x="0" y="39" width="222" height="39"/>
  <SubTexture name="cockpitBlue_0.png" x="586" y="0" width="51" height="75"/>
  <SubTexture name="cockpitBlue_1.png" x="736" y="862" width="40" height="40"/>
  <SubTexture name="cockpitBlue_2.png" x="684" y="67" width="42" height="56"/>
  <SubTexture name="cockpitBlue_3.png" x="336" y="384" width="60" height="61"/>
  <SubTexture name="cockpitBlue_4.png" x="637" y="0" width="47" height="67"/>
  <SubTexture name="cockpitBlue_5.png" x="627" y="144" width="48" height="75"/>
```


Sprite uvs:



$x/\text{image_width}$
 $y/\text{image_height}$

$(x/\text{image_width}) + (\text{width}/\text{image_width}),$
 $y/\text{image_height}$

$x/\text{image_width},$
 $(y/\text{image_height}) + (\text{height}/\text{image_height})$

$(x/\text{image_width}) + (\text{width}/\text{image_width}),$
 $(y/\text{image_height}) + (\text{height}/\text{image_height})$



```
class SheetSprite {
    public:
        SheetSprite();
        SheetSprite(unsigned int textureID, float u, float v, float width, float height, float
size);

        void Draw(ShaderProgram *program);

        float size;
        unsigned int textureID;
        float u;
        float v;
        float width;
        float height;
};

spriteSheetTexture = LoadTexture("sheet.png");

mySprite = SheetSprite(spriteSheetTexture, 425.0f/1024.0f, 468.0f/1024.0f, 93.0f/1024.0f, 84.0f
1024.0f, 0.2);
```



```
void SheetSprite::Draw(ShaderProgram *program) {
    glBindTexture(GL_TEXTURE_2D, textureID);

    GLfloat texCoords[] = {
        u, v+height,
        u+width, v,
        u, v,
        u+width, v,
        u, v+height,
        u+width, v+height
    };

    float aspect = width / height;
    float vertices[] = {
        -0.5f * size * aspect, -0.5f * size,
        0.5f * size * aspect, 0.5f * size,
        -0.5f * size * aspect, 0.5f * size,
        0.5f * size * aspect, 0.5f * size,
        -0.5f * size * aspect, -0.5f * size ,
        0.5f * size * aspect, -0.5f * size};

    // draw our arrays
}

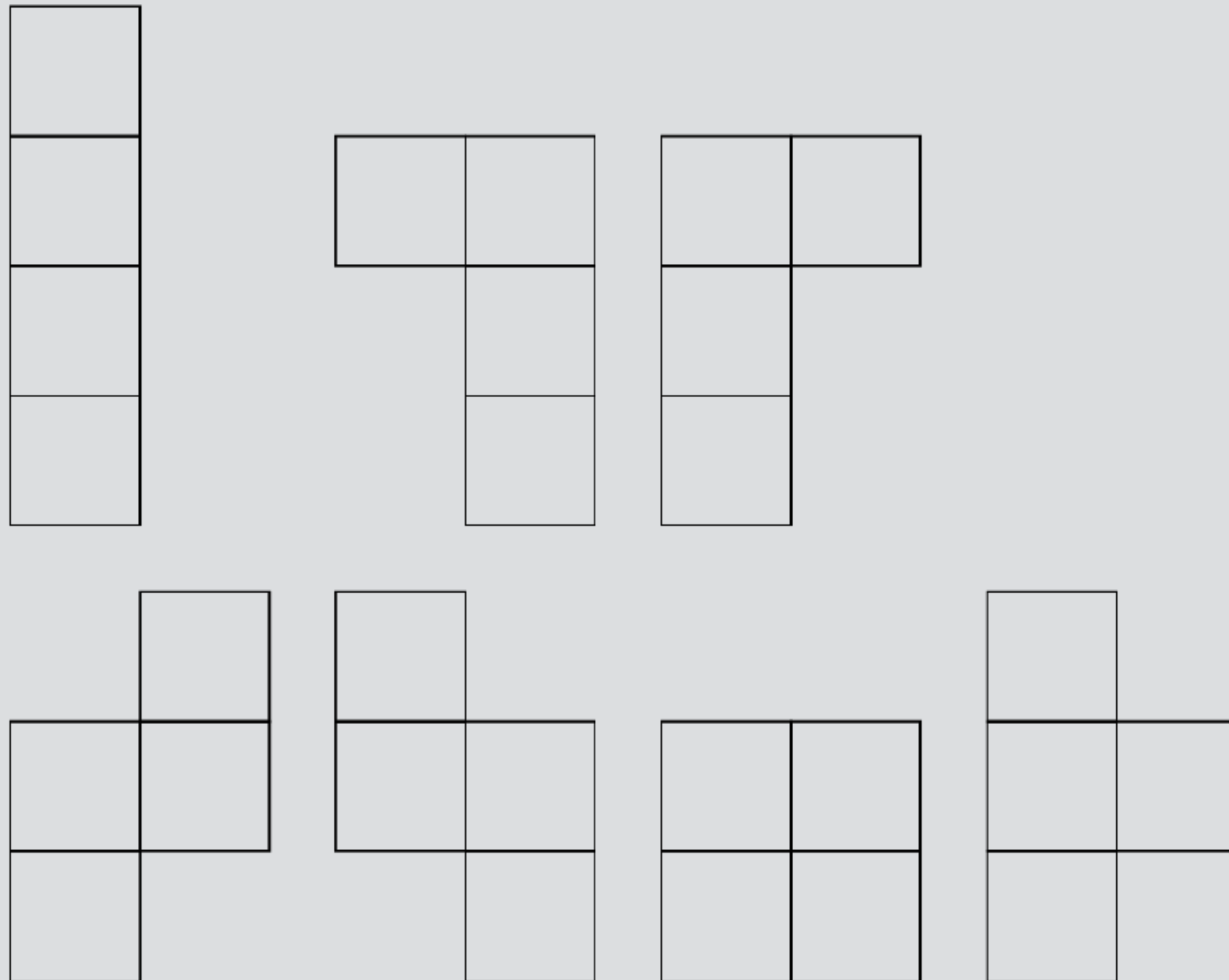
void Render() {
    enemySprite.Draw(program);
}
```

Making your own texture atlases.

Shoebox sprite tool.

<http://renderhjs.net/shoebox/>

Game structure



Managing game objects.

Entities


```
class Vector3 {  
    public:  
  
        Vector3(float x, float y, float z);  
  
        float x;  
        float y;  
        float z;  
  
};
```

A simple vector class will help us keep track of coordinates.

```
class Entity {  
    public:  
  
        void Draw();  
        void Update(float elapsed);  
  
        Vector3 position;  
        Vector3 velocity;  
        Vector3 size;  
  
        float rotation;  
  
        SheetSprite sprite;  
};
```

**Entities are a
useful way
for us to think
about objects
in the game.**

02:46:02 (Normal) (Moderada)

-Paladin: creación finalizada-



NecroSoul: 62945/46116
Alfredo el Grande: 29287/37223
Hernán Cortés: 34548/34548
Ambrosio: 24000/24000
Tsesugi Kenshin: 19438/20043
Kai Kebab: 14495/15424



Onager de asedio

70/70

75+1

0/8

8+1

Teutones

NecroSoul



SCORE<1> 0000 HI-SCORE 0000 SCORE<2>



CREDIT 00

```
std::vector<Entity> entities;
```

```
    Entity myEntity;  
    myEntity.sprite = SheetSprite(spriteSheetTexture, 425.0f/1024.0f, 468.0f/1024.0f,  
93.0f/1024.0f, 84.0f/1024.0f, 0.2);  
    entities.push_back(myEntity);
```

```
void Update(float elapsed) {  
  
    for(int i=0; i < entities.size(); i++) {  
        entities[i].Update(elapsed);  
    }  
}
```

```
void Render() {  
  
    glClear(GL_COLOR_BUFFER_BIT);  
  
    for(int i=0; i < entities.size(); i++) {  
        entities[i].sprite.Draw(program);  
    }  
}
```

Managing dynamically created objects.

HEALTH



POWER

BOSS HEALTH

X8

S

RANK

000.012.400

BULLET



MAGMA

MINE



SHOCK WAVE



CHAIN

178285

ARMS BOMB

320 08

57

1UP=2



CREDITS 09

LEVEL-4

CREDIT 00

Dynamic object creation vs. object pools

Dynamic object creation

Dynamic object creation

- Can be dynamically sized.
- Objects must be manually removed.
- No limit on how many objects can be on the screen.

```
std::vector<Entity> bullets;
```

```
void shootBullet() {  
    Entity newBullet;  
    newBullet.position.x = -1.2f;    // where the bullet starts X  
    newBullet.position.y = 0.0f;    // where the bullet starts Y  
    newBullet.velocity.y = 2.0f;  
    newBullet.timeAlive = 0.0f;  
    bullets.push_back(newBullet);  
}
```

```
bool shouldRemoveBullet(Entity bullet) {  
    if(bullet.timeAlive > 0.4) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

```
bullets.erase(std::remove_if(bullets.begin(), bullets.end(), shouldRemoveBullet), bullets.end());  
  
for(int i=0; i < bullets.size(); i++) {  
    bullets[i].Update(elapsed);  
}
```

Object pools.

Object pools.

- Less prone to memory leaks.
- Have a maximum number of objects.
- Allocated all at once.
- Know how fast things will run with maximum objects.

Object pools.

```
#define MAX_BULLETS 30
int bulletIndex = 0;
Entity bullets[MAX_BULLETS];
for(int i=0; i < MAX_BULLETS; i++) {
    bullets[i].x = -2000.0f;
}
```

```
void shootBullet() {

    bullets[bulletIndex].x = -1.2;
    bullets[bulletIndex].y = 0.0;
    bulletIndex++;
    if(bulletIndex > MAX_BULLETS-1) {
        bulletIndex = 0;
    }
}
```

```
for(int i=0; i < MAX_BULLETS; i++) {
    bullets[i].Update(elapsed);
}
```

Game state and game mode.

Game state.

```
class GameState {  
    public:  
  
        Entity player;  
        Entity enemies[12];  
        Entity bullets[10];  
        int score;  
};
```

```
GameState state;
```

```
void RenderGame(const GameState &game) {  
    // render all the entities in the game  
    // render score and other UI elements  
}
```

```
void UpdateGame(const GameState &game, float elapsed) {  
    // move all the entities based on time elapsed and their velocity  
}
```

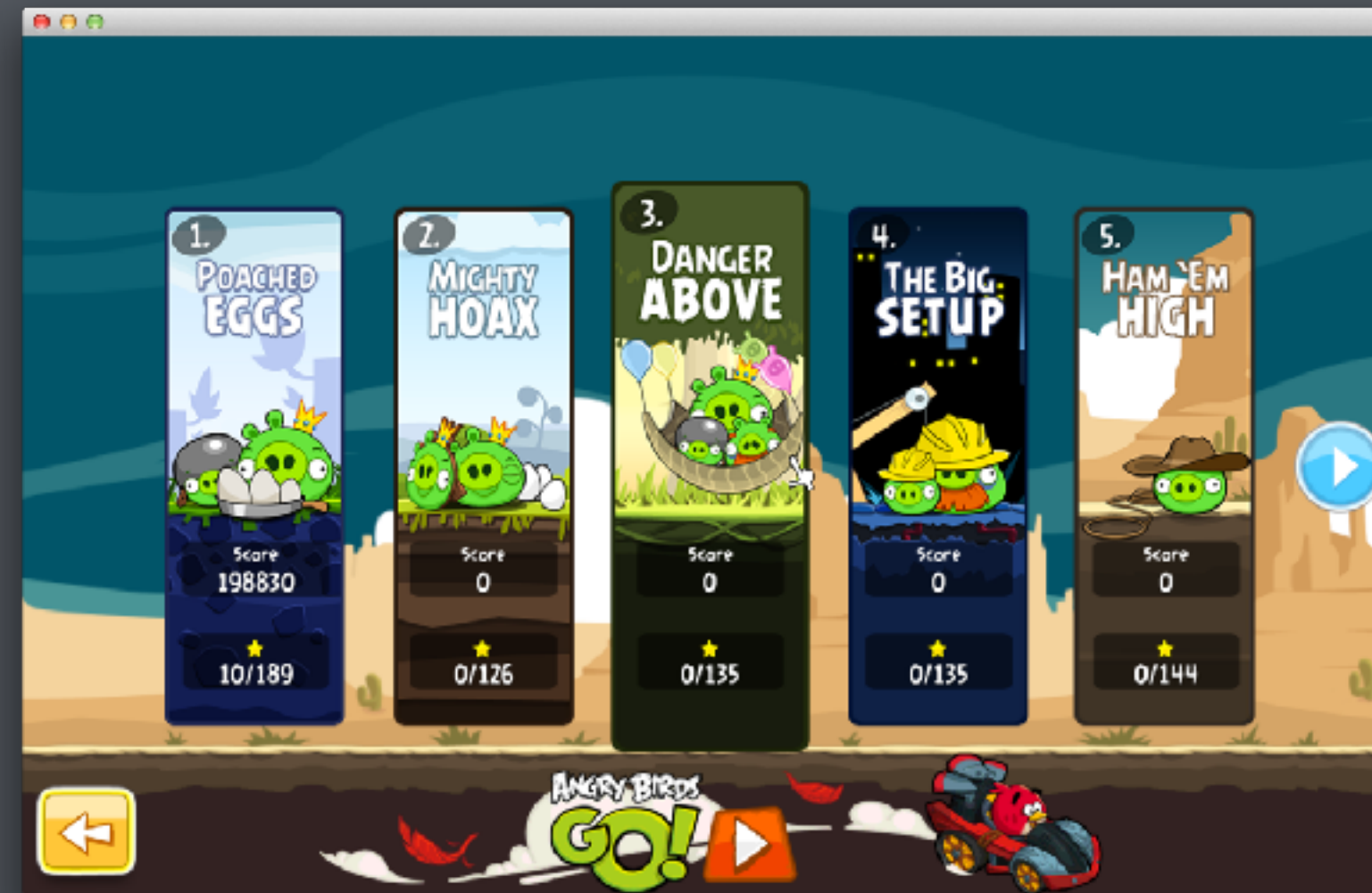
```
void ProcessInput(const GameState &game) {  
}
```

Game mode.

Main menu



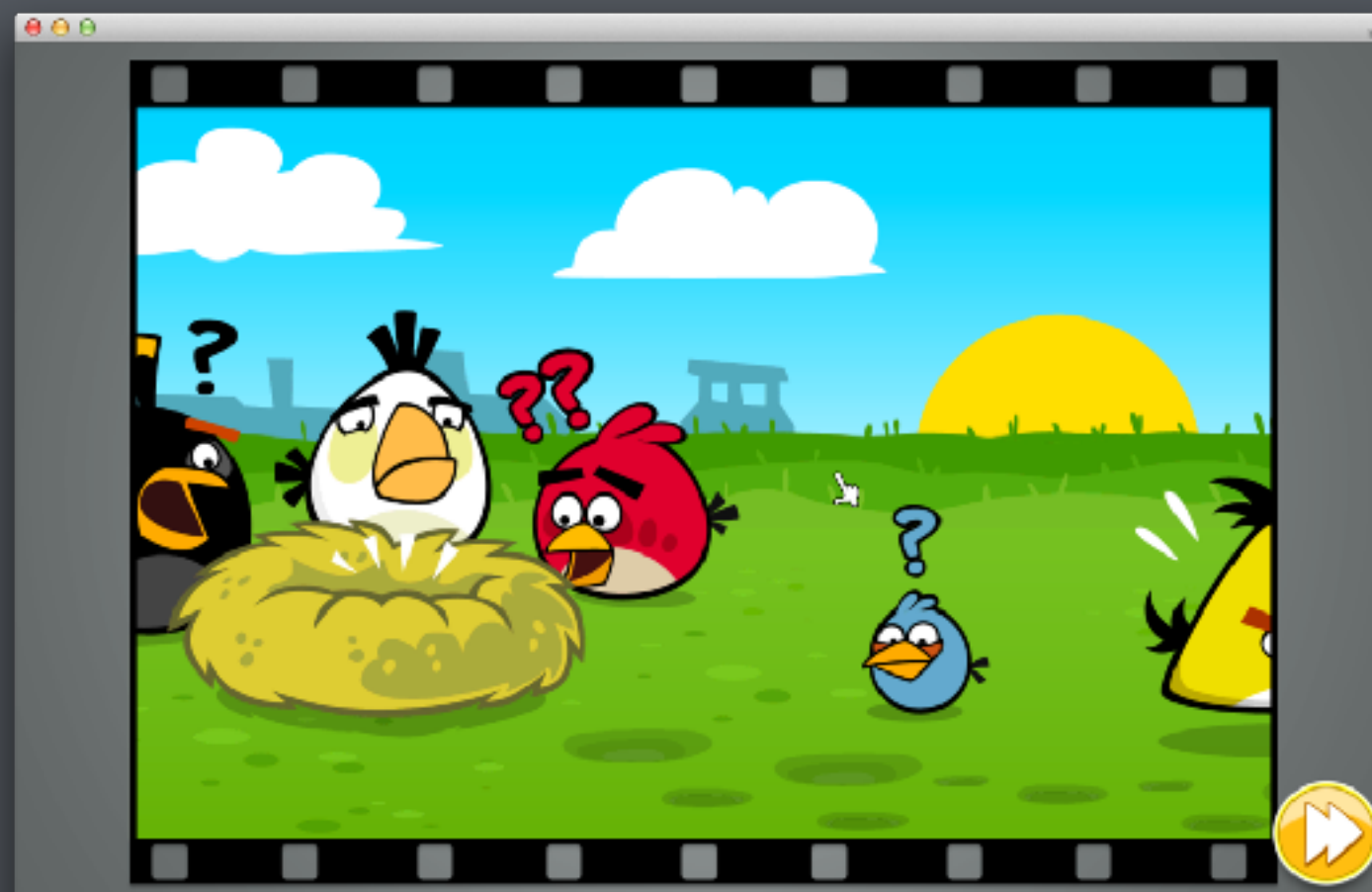
Chapter select



Level select



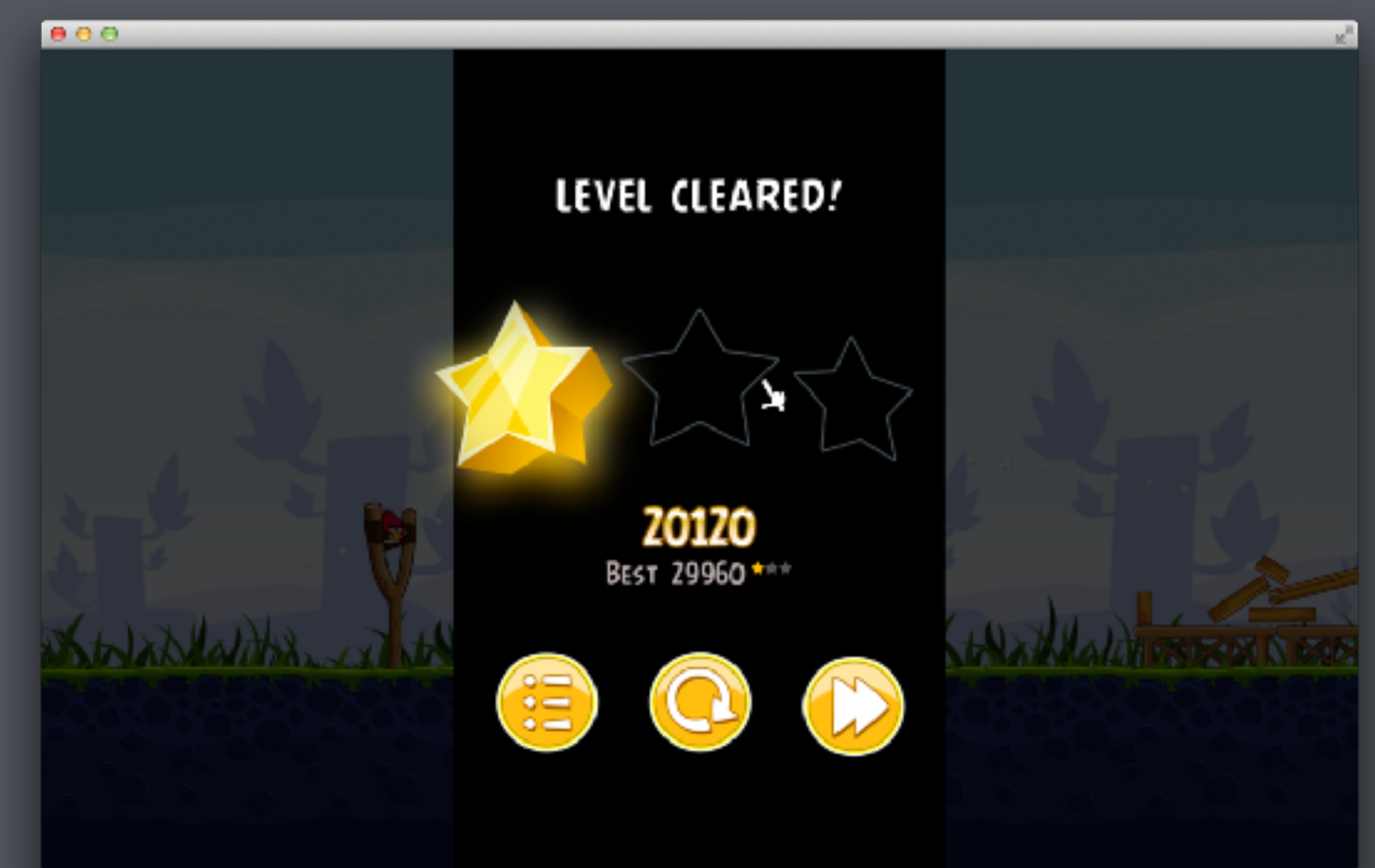
Cutscene



Game level



Win screen



```
enum GameMode { STATE_MAIN_MENU, STATE_GAME_LEVEL, STATE_GAME_OVER};
```

```
GameMode mode;
```

```
GameState state;
```

```
void Render() {
```

```
    switch(mode) {
```

```
        case STATE_MAIN_MENU:
```

```
            RenderMainMenu();
```

```
        break;
```

```
        case STATE_GAME_LEVEL:
```

```
            RenderGameLevel(state);
```

```
        break;
```

```
    }
```

```
}
```

```
void Update(float elapsed) {
```

```
    switch(mode) {
```

```
        case STATE_MAIN_MENU:
```

```
            UpdateMainMenu(elapsed);
```

```
        break;
```

```
        case STATE_GAME_LEVEL:
```

```
            UpdateGameLevel(state, elapsed);
```

```
        break;
```

```
    }
```

```
}
```

```
void ProcessInput() {
```

```
    switch(mode) {
```

```
        case STATE_MAIN_MENU:
```

```
            ProcessMainMenuInput();
```

```
        break;
```

```
        case STATE_GAME_LEVEL:
```

```
            ProcessGameLevelInput(state);
```

```
        break;
```

```
    }
```

```
}
```



```
enum GameMode { STATE_MAIN_MENU, STATE_GAME_LEVEL, STATE_GAME_OVER};
```

```
GameMode mode;
```

```
MainMenu mainMenu;
```

```
GameLevel gameLevel;
```

```
void Render() {
```

```
    switch(mode) {
```

```
        case STATE_MAIN_MENU:
```

```
            mainMenu.Render();
```

```
        break;
```

```
        case STATE_GAME_LEVEL:
```

```
            gameLevel.Render();
```

```
        break;
```

```
    }
```

```
}
```

```
void Update(float elapsed) {
```

```
    switch(mode) {
```

```
        case STATE_MAIN_MENU:
```

```
            mainMenu.Update(elapsed);
```

```
        break;
```

```
        case STATE_GAME_LEVEL:
```

```
            gameLevel.Update(elapsed);
```

```
        break;
```

```
    }
```

```
}
```

```
void ProcessInput() {
```

```
    switch(mode) {
```

```
        case STATE_MAIN_MENU:
```

```
            mainMenu.ProcessInput();
```

```
        break;
```

```
        case STATE_GAME_LEVEL:
```

```
            gameLevel.ProcessInput();
```

```
        break;
```

```
    }
```

```
}
```



Space Invaders

<https://www.youtube.com/watch?v=axlx3o0codc>

Assignment



- Make Space Invaders
- It must have 2 game modes: TITLE SCREEN and GAME LEVEL and use a game state.
- It must display text
- It must use sprite sheets (uniform or non)
- You can use any graphics you want (it doesn't have to be in space! :)