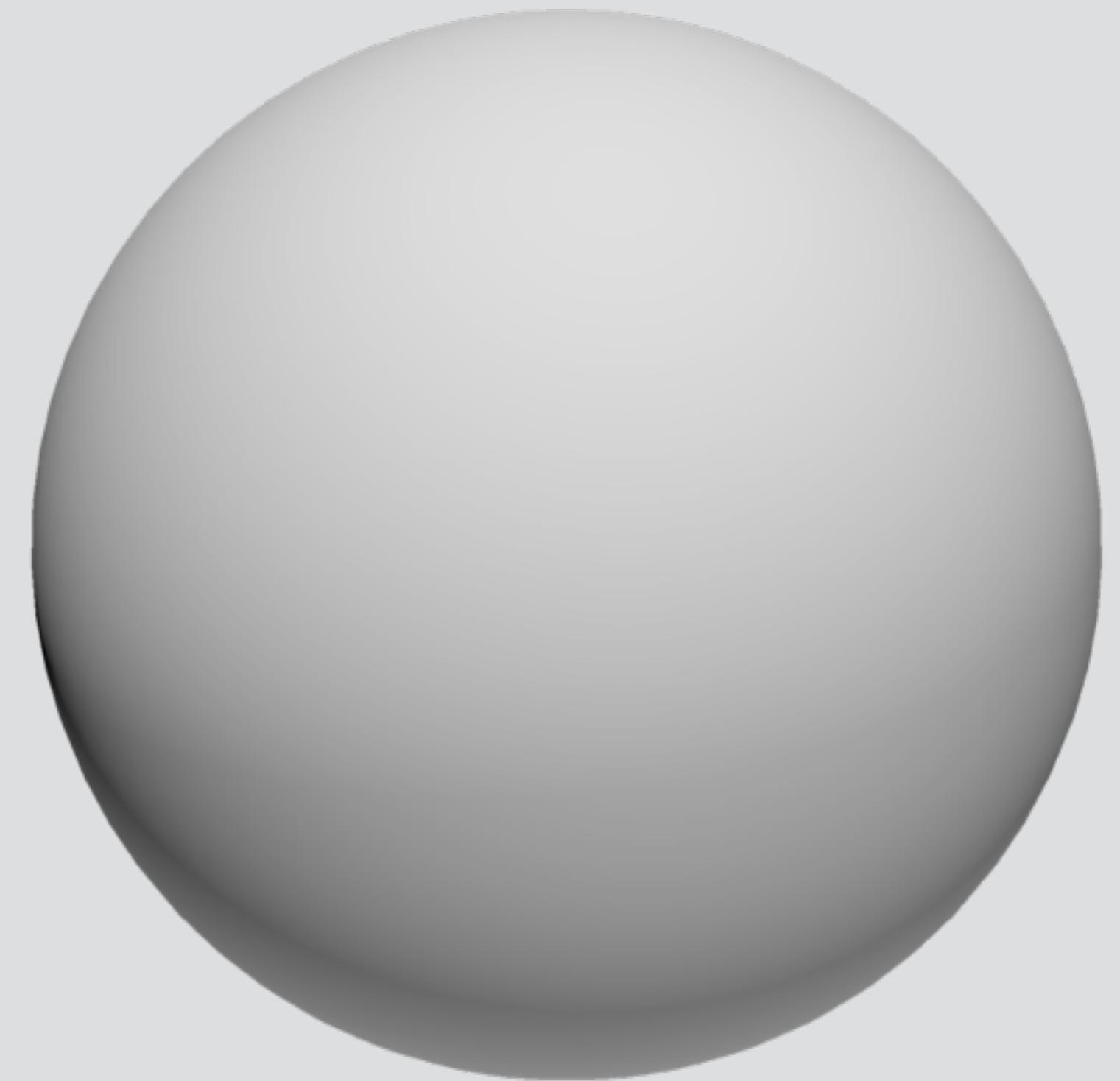


Graphics Foundations



Part 3.5

Non-uniform sprite sheets.

Uniform





Non-uniform

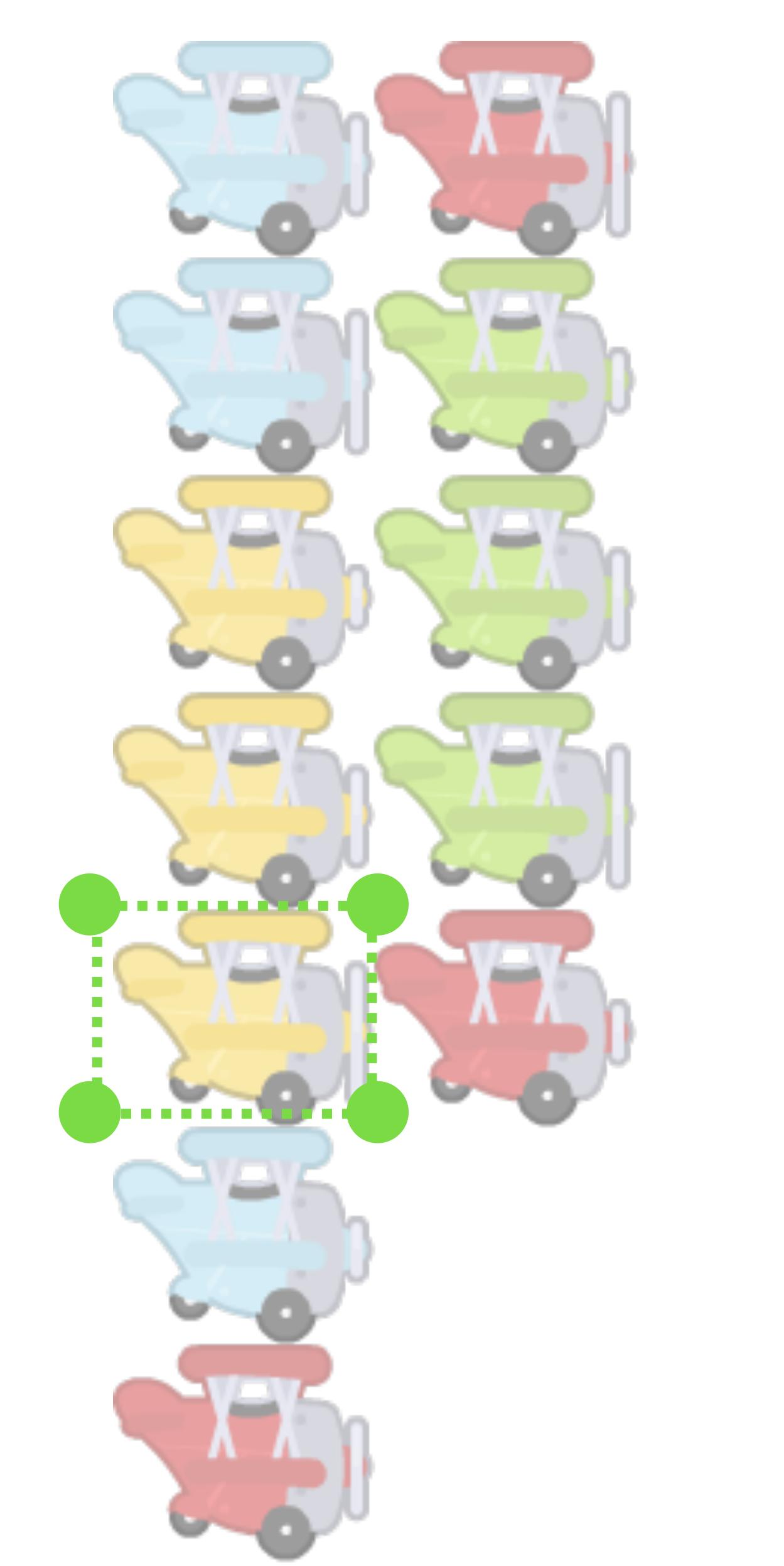


Need to keep a **list of coordinates** for non-uniform sprite sheets.

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();  
  
    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() {
    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
}

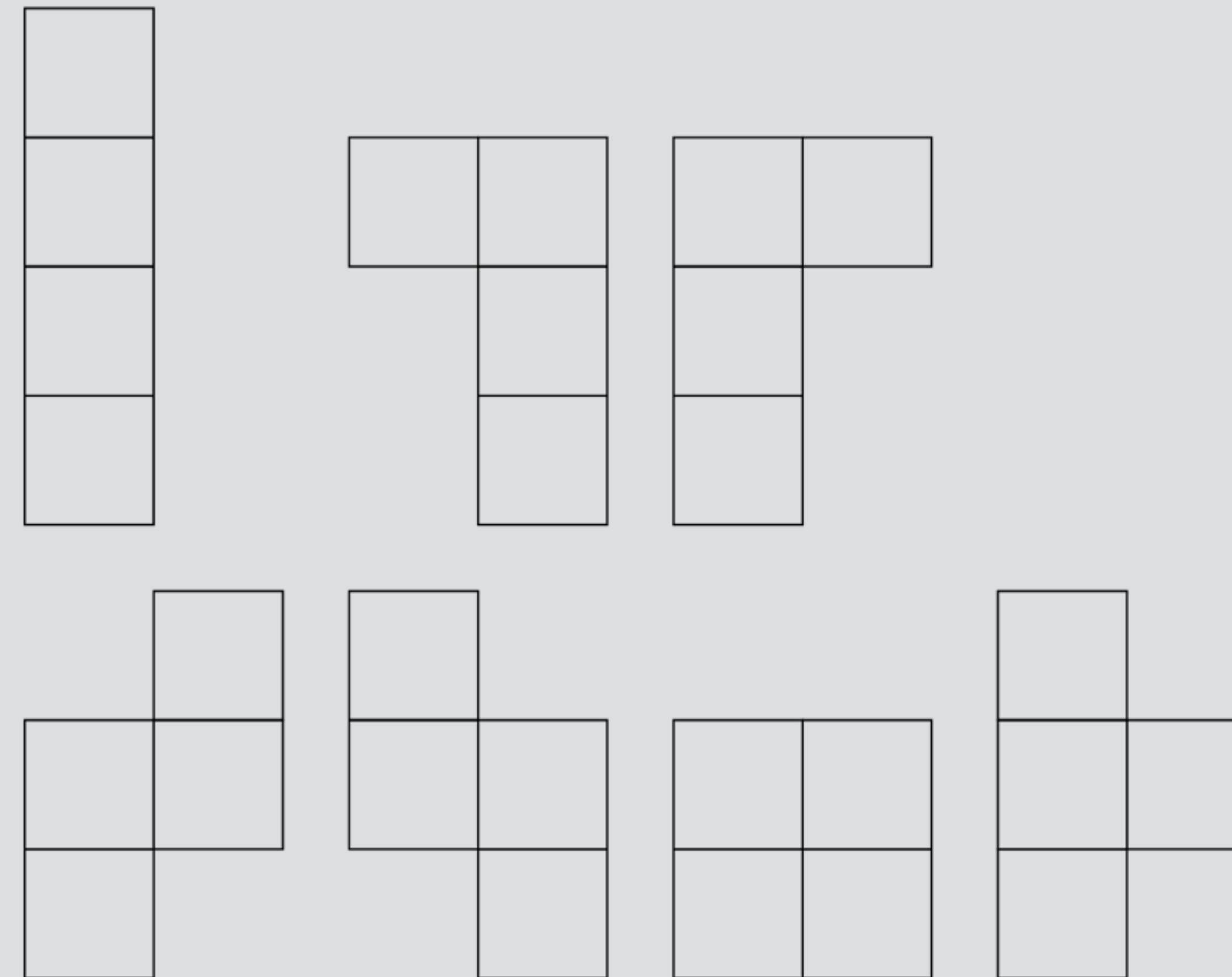
void ClassDemoApp::Render() {
    enemySprite.Draw();
}
```

Making your own texture atlases.

Shoebox sprite tool.

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

Game structure



Managing game objects.



Onager de asedio
70/70
75+1
0/8
8+1

Teutones
NecroSoul

CONQUERORS



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



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 ClassDemoApp::Update(float elapsed) {

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

```
void ClassDemoApp::Render() {

    glClear(GL_COLOR_BUFFER_BIT);

    for(int i=0; i < entities.size(); i++) {
        entities[i].Render();
    }
}
```

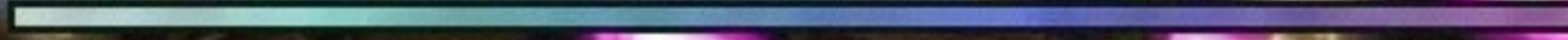
Managing dynamic objects.



HEALTH



POWER



BOSS HEALTH



x8

S

RANK

000.012.400



BULLET

SHOCK WAVE

MAGMA

MINE



CHAIN

178285



ARMS BOMBS
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<Bullet> bullets;
```

```
void shootBullet() {
    Bullet newBullet;
    newBullet.x = -1.2;
    newBullet.y = 0.0;
    newBullet.angle = (float)(45 - (rand() % 90));
    newBullet.speed = 2.0;
    bullets.push_back(newBullet);
}
```

```
bool shouldRemoveBullet(Bullet 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;
Bullet bullets[MAX_BULLETS];
for(int i=0; i < MAX_BULLETS; i++) {
    bullets[i].x = -2000.0f;
}
```

```
void ClassDemoApp::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 states.

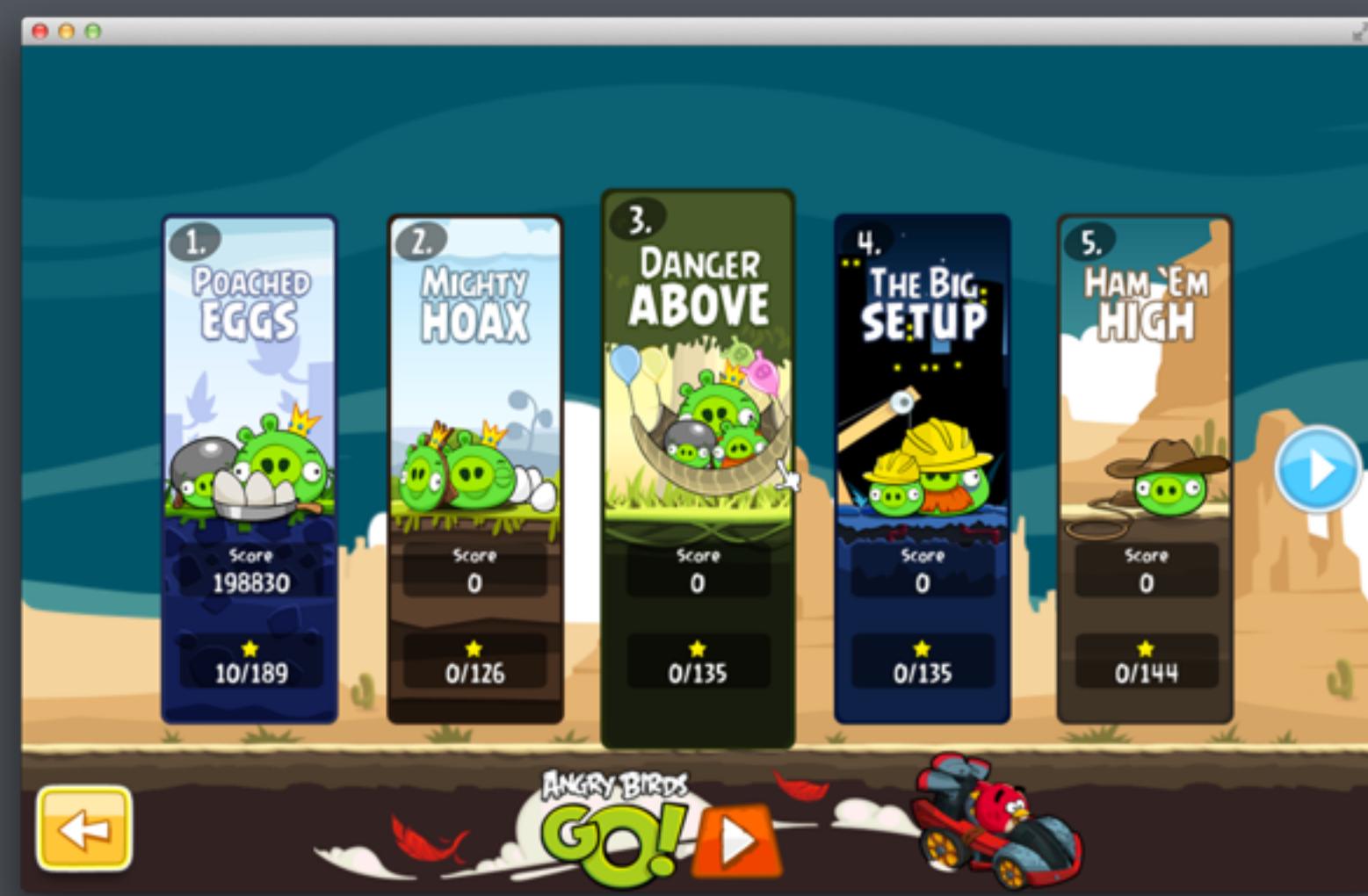
Main menu



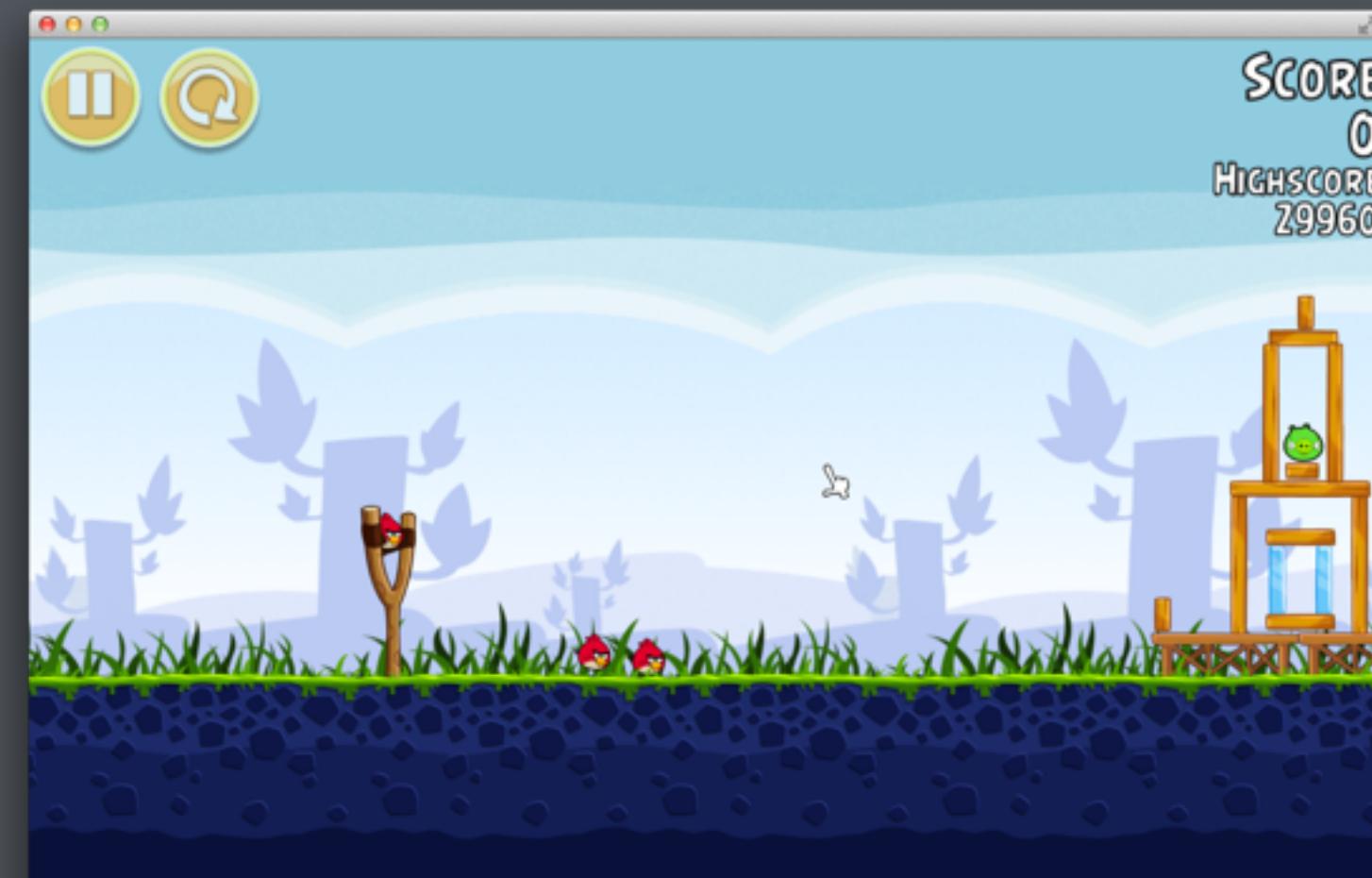
Cutscene



Chapter select



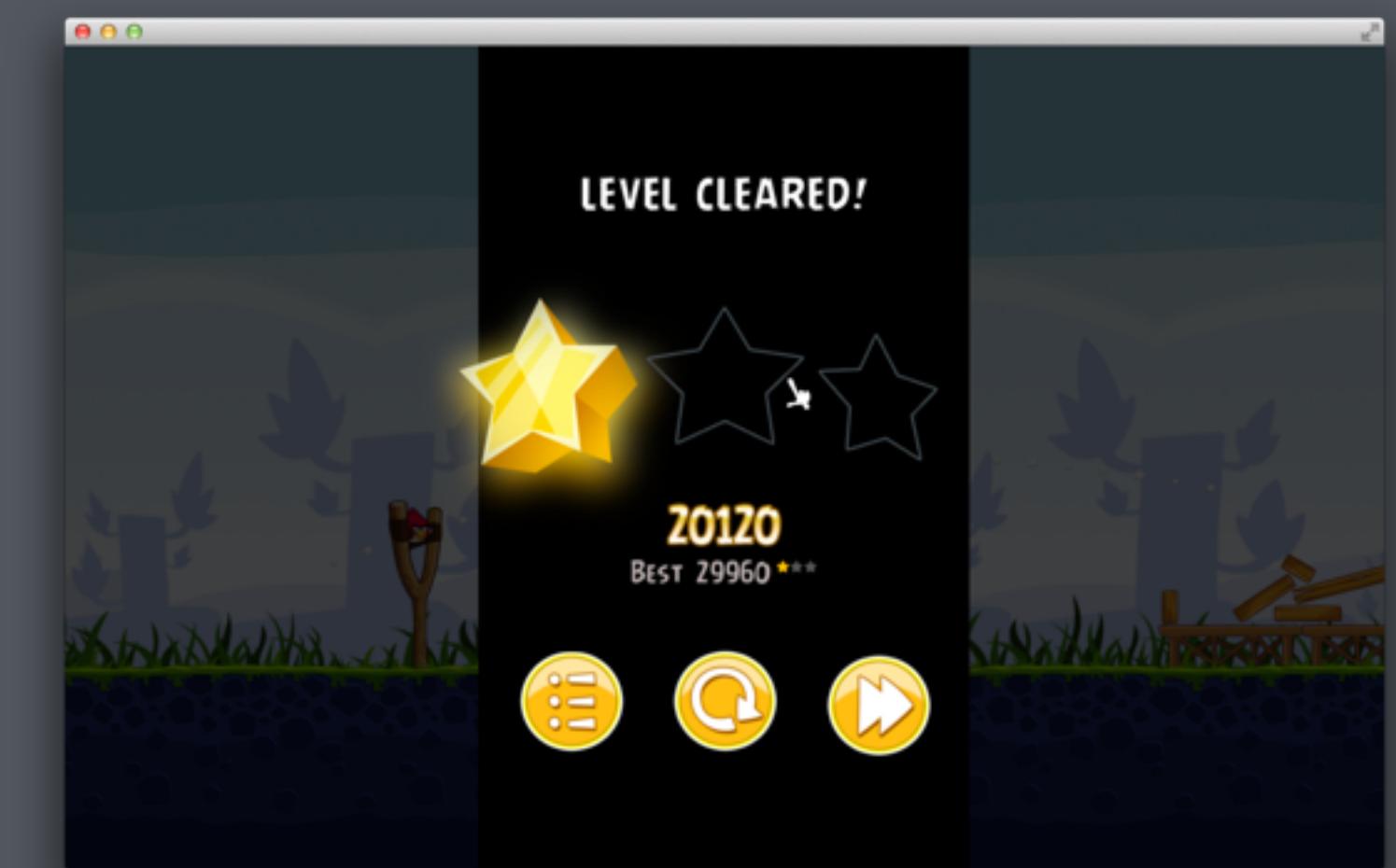
Game level



Level select



Win screen



```
enum GameState { STATE_MAIN_MENU, STATE_GAME_LEVEL };

int state;

void ClassDemoApp::Render() {
    switch(state) {
        case STATE_MAIN_MENU:
            RenderMainMenu();
        break;
        case STATE_GAME_LEVEL:
            RenderGameLevel();
        break;
    }
}

void ClassDemoApp::Update() {
    switch(state) {
        case STATE_MAIN_MENU:
            UpdateMainMenu();
        break;
        case STATE_GAME_LEVEL:
            UpdateGameLevel();
        break;
    }
}
```

```
enum GameState { STATE_MAIN_MENU, STATE_GAME_LEVEL };

int state;

void ClassDemoApp::Render() {
    switch(state) {
        case STATE_MAIN_MENU:
            mainWindow.Render();
        break;
        case STATE_GAME_LEVEL:
            gameLevel.Render();
        break;
    }
}

void ClassDemoApp::Update() {
    switch(state) {
        case STATE_MAIN_MENU:
            mainWindow.Update();
        break;
        case STATE_GAME_LEVEL:
            gameLevel.Update();
        break;
    }
}
```



Space Invaders

https://www.youtube.com/watch?v=437Ld_rKM2s

Assignment

- ▶ Make Space Invaders
- ▶ It must have 2 states: TITLE SCREEN, GAME
- ▶ It must display text
- ▶ It must use sprite sheets
- ▶ You can use any graphics you want (it doesn't have to be in space! :)