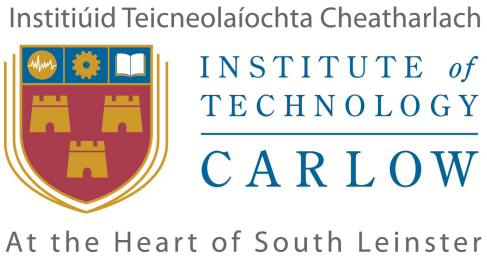
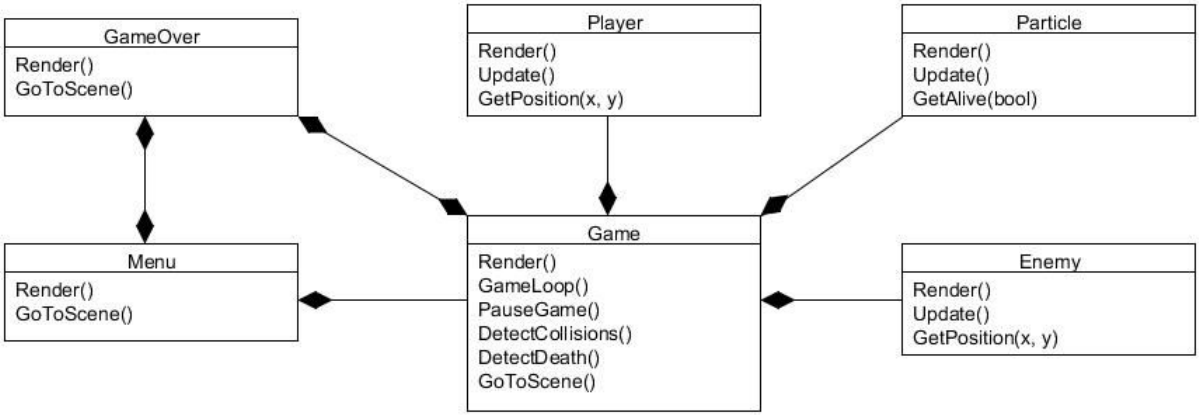


Computer Games Development
CW208 - TDD
Year III

Box Hopper

Evan Cunningham - C00179980

Architecture



Approaches

User Interface; Conveying information to the Player

Approach 1

Draw a health number to the screen which displays how many more hits the player can take before being killed

Pros: Simple, easy to understand

Cons: Distracting, ugly

Approach 2

Draw a health bar to the screen which displays how many more hits the player can take before being killed

Pros: Used in many games so should be quickly understood

Cons: Involves multiple Images being loaded, slow, outdated

Approach 3

Scrap the health approach altogether and simply have the player be shown exactly how close they are to death by their proximity to the approaching skull

Pros: Intuitive, minimalistic

Cons: Not as easy to gauge exactly how many more hits you can take

Collisions

Approach 1

Use the built in Cocos box collision or similar built in feature

Pros: Most of the leg work is done

Cons: I don't understand the system inside and out and if a problem arises it will be more difficult to attempt to fix, also only works for 1 version of the game

Approach 2

Use my own collision detection

Pros: I understand the system in its entirety

Cons: Time consuming, maybe not as streamlines as built in system

Feature Design

Feature 1: Scenes and Scenemanager

Task 1: Create Scene Manager

- AddScene()
    - Add Scene creates a new scene and adds it to a list of scenes in the scenemanager
- GoToScene()
    - Go To Scene makes the newly created scene the active scene

Task 2: Handle Inputs

- addEventListener(mousedown)
    - This waits for a mousedown event (player clicking, or tapping the screen)
- input(x,y)
    - This sends the mouse / tap location to the main class, which sends it to the scenemanager, which then sends it into the currently selected scene

Task 3: Create and Draw Buttons which know if they're pressed

- InputHandler()
    - The Scenes will pass the button class an event handling method with the location of the event
- ButtonCreate()
    - This creates a button with desired parameters
- Renderer()
    - This draws the button, along with everything else
- Button.IsClicked(x,y)

    - This is the method in the button which will check if the input was within the bounds of the button

Feature 2: Player and GameScene

Task 1: Create, and Draw, Scene and Player

- SceneManager.AddScene(GameScene)
- GameScene.createPlayer(pos)
- GameScene.Render()
    - Player.Render()

Task 2: Player Logic

- input(x,y)
    - Player.Input(x,y)
        - This will let the player know that an event has happened in order for it to carry out its function (jump)
- game.Render()
    - player.Render()
- Game.Update()
    - Player.Update()

Feature 3: Enemies

Task 1: Create Enemies

- Game.CreateEnemies()
- Game.Update()
    - Enemy.Update()
- Game.Render()
    - Enemy.Render()

Task 2: Collision Detection

- Game.DetectCollisions(enemies, player)
    - This is passed the list of enemies and the player object, it then checks if any of the enemies are intersecting with the player
- if(DetectCollisions) Game.PlayerHit()

    - If the detect collisions method detects a hit, the player will react accordingly

Task 3: Destroying Enemies

- Game.EnemyOutOfBounds(enemies)
    - This function is passed a list of enemies, and checks if any enemies have left the bounds of the screen
- For(Enemies[]) if(EnemyOutOfBounds) ~Enemies[i]
    - If they're off screen, they are removed

Feature 4: Death and Game Over

Task 1: Create GameOverScene

- Game.CreateGameOver()
- GameOver.Render()
    - GameOverButtons.Render()

Task 2: Create Death

- Game.CreateDoom()

Task 2: Detect Death

- Game.DetectDeath(doom, player)
    - This is detect collisions, but detects collisions between the singular objects doom and player
- if(DetectDeath) Game.GoToScene(gameOver)
    - If the player hits the kill box, the player is sent to the GameOver Scene

Feature 5: Animation

Task 1: Animate Player

- playerSprite.Animate()
    - In cocos there is a built in animation method, sprite->runAnimation(), and in JS this is done drawing subsets of a spritesheet

Task 2: Animate Doom

- doom.runAnimation()

Feature 6: Audio

Task 1: Background Music

- Audio.playBackgroundMusic()

Task 2: Play Sound Effects

- Game.DetectCollisions()
    - Audio.playSoundEffect()
        - In Cocos there is a built in Audio system with both of these functions

Feature 7: Particle System

Task 1: Create Particles

- Game.CreateParticles(playerPos)

Task 2: Update Particles

- Game.UpdateParticles(dt)

Task 3: Kill Particles

- For(Particles[]) getParticleAlive() ~Particle[i]
    - Particle class has a get method for the bool isAlive, if the particle is alive for it's lifespan (A randomly created number within set parameters) this bool is set to false. Particle is then removed.

Feature 8: Tutorial System

Task 1: Create Tutorial

- Renderer.drawTutorial()

Task 2: End Tutorial

- Game.TutorialOver()
    - There will be certain conditions which must be met in order to finish the tutorial
- Renderer.drawTutorial(NULL)

Feature 9: Score & UI

Task 1: Track Score

- Game.score

Task 2: Draw UI

- Renderer.DrawScore()

Feature 10: Option

Task 1: Create Option Button

- buttons[].add(OptionButton)

Task 2: Create Optional Feature

- Bool Option;

Task 3: Change Optional Feature

- optionButton.IsClicked(x,y)
    - Option = !Option;