Tuffy Track

Created with Godot

Jennifer Felton

California State University, Fullerton

CPSC 254, Software Development with Open Source Software

Professor David Heckathorn

May 2, 2020

Overview:

Tuffy track is a 3D game created using the Godot game engine. The goal of the game is to collect the coins without being hit by the enemies.

Godot Game Engine:

The Godot game engine was created in 2007 by Juan Linietsky and Ariel Manzur. It is cross platform and works on Windows, Linux and Mac. It is an Open Source game engine, and anyone can view their github and contribute to the project. Games can be created for PC, mobile and web platforms. Unlike Unity, Developers who create games with Godot never have to pay Godot a portion of the game proceeds. Godot is free to use and uses a MIT license. As Godot says on their website, “Your game is yours, down to the last line of engine code”

Godot games are created using scenes. Each aspect of the game is first created as an individual scene, then combined together into a main scene. There are many nodes that make it easy to build any game by stacking the parts you need. No outside assets are needed. If you need more complicated game functions, it’s easy to create your own custom tools and nodes. It supports different programming languages like GDScript, C#, C++, Visual Scripting, Python and more, with a built in editor. It’s easy for a beginner to learn, yet has powerful tools for experienced game Developers.

Godot has it’s own coding language called GDScript, which is a python based language. All variables do not need to be defined by type, just simply as a var. Functions are simple and easy to write, with no complicated brace system. In the built in editor, suggestions are shown to help you choose the right functions for your project. GDScript has well written documentation and is simple to learn to use.

Tuffy Track Creation Process:

Tuffy Track was originally designed to be a go-kart like 3D game. The game was themed around Cal State Fullerton. The player would drive around the track as Tuffy, the college mascot, and collect financial aid. Financial Aid was represented by coins. Professors would get in the way and if you came into contact with one, the game would end and you would lose a life. Professors would be removed from the track by throwing projects at them. The game would be

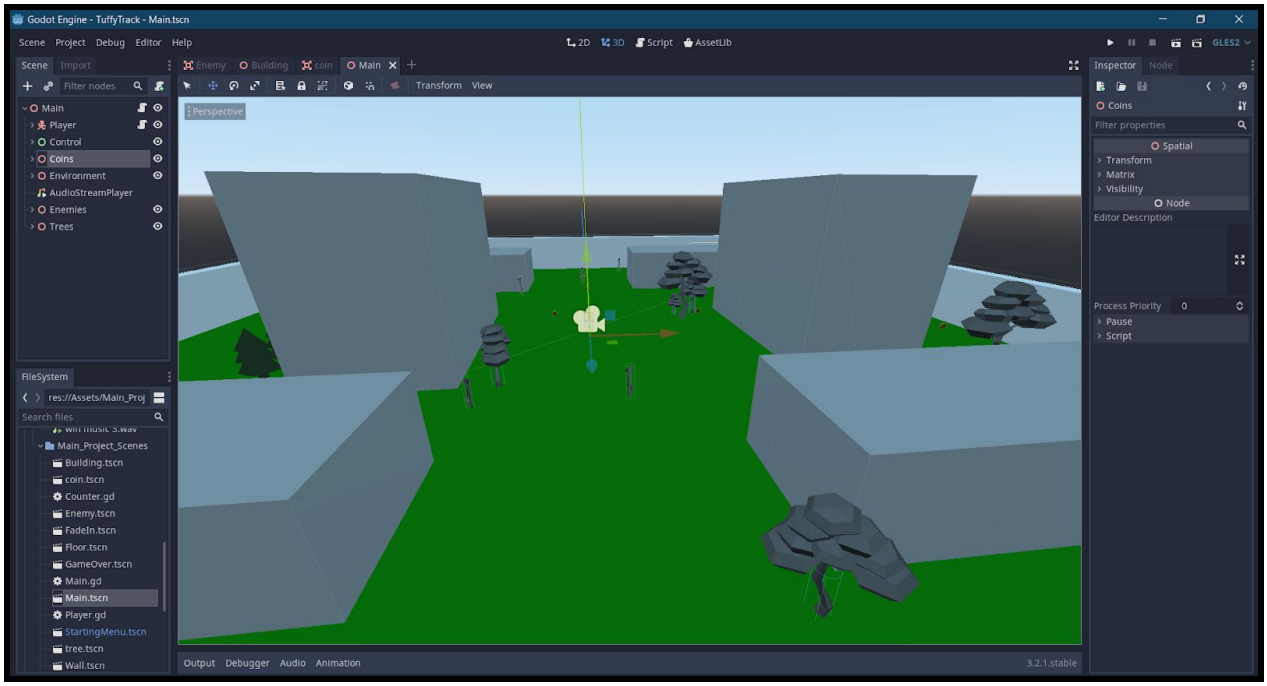
over either when the financial aid was collected, or when the timer ran out. Unfortunately, due to time and pandemic considerations, the game was pared down into its base features.

Some of the difficulties we ran into was the amount of time it would take to create our needed game assets. We could not create an acceptable looking Tuffy, GoKart, buildings and environmental aspects in GoDot itself, so we would have to create them from scratch in Blender and import them. This would require us learning art, and Blender, so we decided to put that aside for if we had time left over. Blender is an Open Source 3D computer graphic software, and we did use it to import some of the game assets. We eventually learned about a resource called Open Game Art that included free game development assets along with licenses for use. The enemies, trees and music all came from Open Game Art.

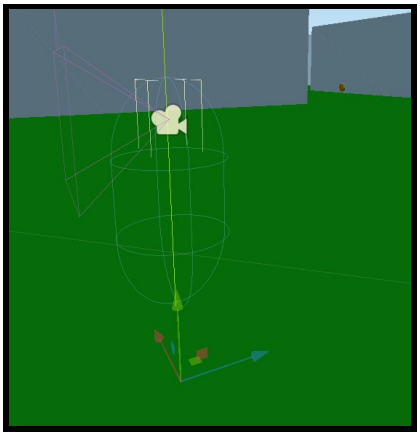
Tuffy Track Features:

The video of the game in its “turn in state” can be found here: https://youtu.be/PpVLFv9TMiA

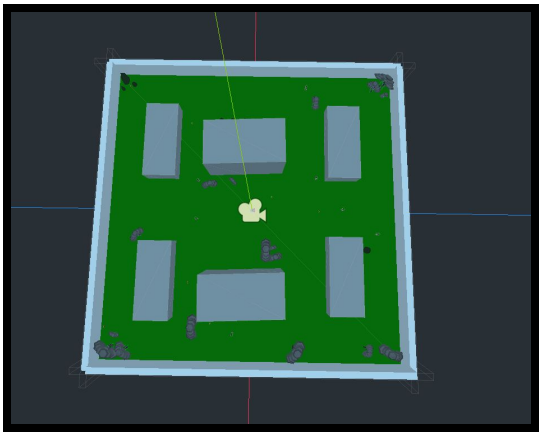
The game in its “turn in state” includes a first person custom coded camera and player controller, buildings, trees, enemies, coins and menus.



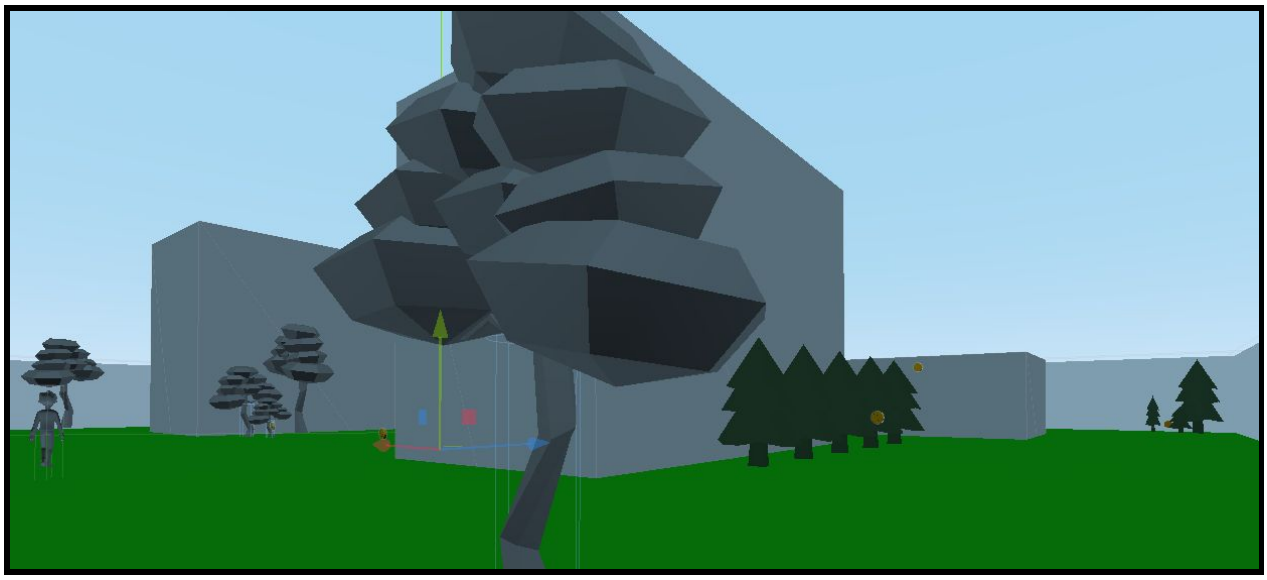
The player and camera were coded for this game specifically. When the player selects New Game, the game engine captures control of the mouse. This allows the player to use the mouse to look around the environment. This provides a 360 degree view. The player uses WASD to move around the environment and the spacebar to jump. Pressing E provides a speed boost. Esc allows the player to leave the game.



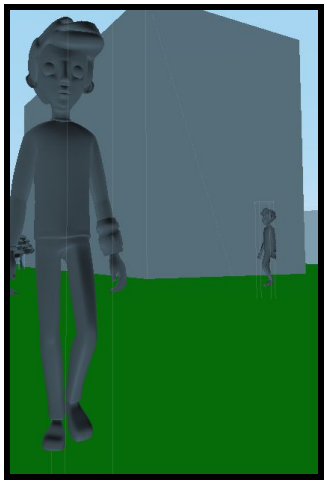
The floor, walls, and buildings were all created as individual scenes in Godot and then put together in the main scene. They are made up of a Mesh Instance, Static Body, and Collision shape, so they can not be passed through. They are basic shapes to keep the player in the playing zone and provide a challenge. The building layout is suggestive of the Computer Science and Engineering buildings on campus.



The trees in the environment were downloaded from OpenGameArt and imported using Blender. Each type of tree was placed in its own scene, given a mesh instance, static body and collision shape. After that, each tree was duplicated from the master tree, individually placed in its location and resized appropriately.



The enemies were based on a design called Gilbert from OpenGameArt. Gilbert came with his own walking animation. Each Enemy was duplicated into the scene and given a unique walking animation path. Each direction in the walking path must be individually plotted for each axis. There is a script that handles the “game over” trigger if the player enters the Enemy’s area.

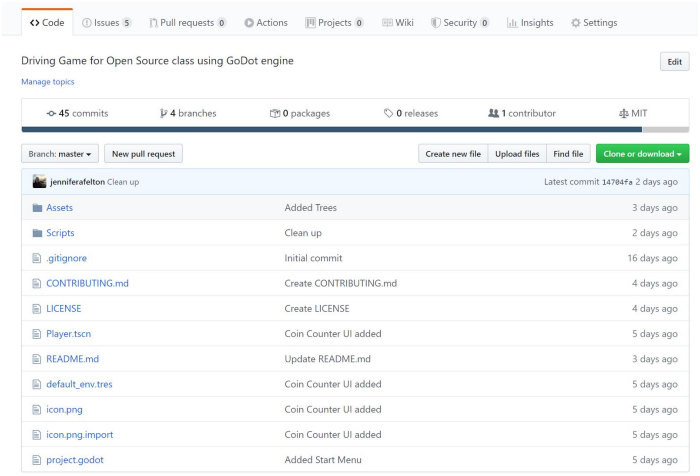


The coins are based on a design by BornCG. He provides a coin blender object that has a matching coin .png file. The coin blender object was used for our coins, and coin removal on area enter was coded by us. The coin count is updated with a script and displayed with the coin .png on the corner of the screen.

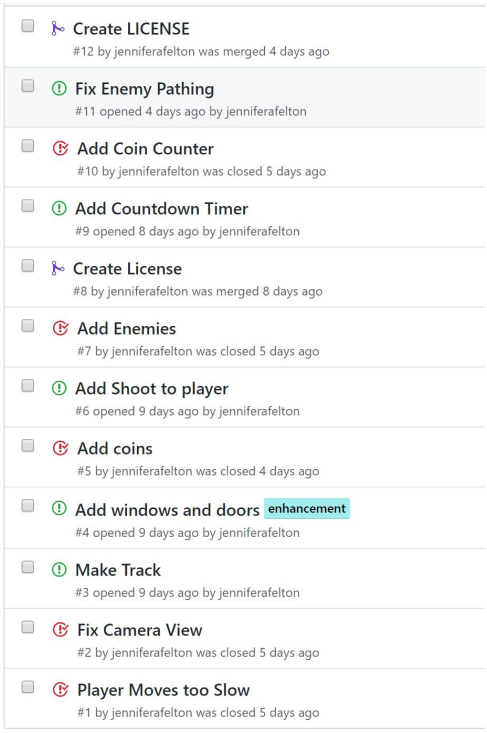
The menus were created using GoDot's 2D container system. The opening menu contains a button with a script. The Game Over and You Won screens are both a giant button that has a script to restart the game.

CASE Tools:

The github repository can be found here: https://github.com/jenniferafelton/TuffyTrack
Github was used to manage game updates and additions. It was used to share game assets.



The Issues tab was used as a ToDo list of features, and as features were added or problems were encountered, they were closed or commented appropriately. There are still outstanding issues that may be fixed at a later time.



The github repository also has the MIT License, contributing guidelines, and acknowledgements. This allows for others to contribute to the project if they wish.

Code:

Our Code was written in GDScript. Below are the scripts we wrote for Tuffy Track.

Main.gd is used when the main gameplay scene loads:

```
extends Spatial

# Called when the node enters the scene tree for the first time.
func _ready():
	Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED)

# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
	if (Input.is_action_just_pressed("ui_cancel")):
		Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)
		get_tree().quit()
	if(Input.is_action_just_pressed("restart")):
		get_tree().reload_current_scene()
```

Player.gd is the controller for the camera and the player:

```
extends KinematicBody

var camera_angle = 0
var mouse_sensitivity = 0.3

var velocity = Vector3()
var direction = Vector3()

# fly variables
const FLY_SPEED = 40
const FLY_ACCEL = 4

# walk variables
var gravity = -9.8 * 3
const MAX_SPEED = 20
const MAX_RUNNING_SPEED = 30
const ACCEL =2
const DEACCEL = 6

# jumping
var jump_height = 15

# Called when the node enters the scene tree for the first time.
```

```
func _ready():
	pass

func _physics_process(delta):
	walk(delta)

func _input(event):
	if event is InputEventMouseMotion:
		$Head.rotate_y(deg2rad(-event.relative.x * mouse_sensitivity))

		var change = -event.relative.y * mouse_sensitivity
		if change + camera_angle < 90 and change + camera_angle >= -90:
			$Head/Camera.rotate_x(deg2rad(change))
			camera_angle += change


func walk(delta):
	direction = Vector3()

	# get the rotation of the camera
	var aim = $Head/Camera.get_global_transform().basis

	# check input and change direction
	if Input.is_action_pressed("move_forward"):
		direction -= aim.z
	if Input.is_action_pressed("move_backward"):
		direction += aim.z
	if Input.is_action_pressed("move_left"):
		direction -= aim.z
	if Input.is_action_pressed("move_right"):
		direction += aim.z

	# I dont know why this direction doesn't work, commented it out
	#direction = direction.nomralized()

	velocity.y += gravity * delta

	var temp_velocity = velocity
	temp_velocity.y = 0

	var speed
	if Input.is_action_pressed("move_sprint"):
```

```
            speed = MAX_RUNNING_SPEED
      else:
            speed = MAX_SPEED

      # where would the player go at max speed
      var target = direction * speed

      var accelteration
      if direction.dot(temp_velocity) > 0:
            accelteration = ACCEL
      else:
            accelteration = DEACCEL

      # calculate a portion of the distance to go
      temp_velocity = velocity.linear_interpolate(target, accelteration *
delta)

      velocity.x = temp_velocity.x
      velocity.z = temp_velocity.z
      # move
      velocity = move_and_slide(velocity, Vector3(0,1,0))

      if Input.is_action_just_pressed("jump"):
            velocity.y = jump_height



func fly(delta):
      direction = Vector3()

      # get the rotation of the camera
      var aim = $Head/Camera.get_global_transform().basis

      # check input and change direction
      if Input.is_action_pressed("move_forward"):
            direction -= aim.z
      if Input.is_action_pressed("move_backward"):
            direction += aim.z
      if Input.is_action_pressed("move_left"):
            direction -= aim.z
      if Input.is_action_pressed("move_right"):
            direction += aim.z
```

```
	#direction = direction.nomralized()

	# where would the player go at max speed
	var target = direction * FLY_SPEED

	# calculate a portion of the distance to go
	velocity = velocity.linear_interpolate(target, FLY_ACCEL * delta)

	# move
	move_and_slide(velocity)


func _on_Enemy_body_entered(body):
	if body.name == "Player":

get_tree().change_scene("res://Assets/Main_Project_Scenes/GameOver.tscn")
```

Counter.gd displays and manages the coin counter and it's UI:

```
extends Label

var coins = 0

func _ready():
	text = String(coins)

func _on_coin_coinCollected():
	coins = coins + 1
	_ready()
	if coins == 10:
		$Timer.start()

func _on_Timer_timeout():
	get_tree().change_scene("res://Assets/Main_Project_Scenes/You
Win.tscn")
```

Coin.gd handles coin animation and removal on player enter:

```
extends Area
```

```
signal coinCollected

func _ready():
	pass

func _physics_process(delta):
	rotate_y(deg2rad(3))

func _on_coin_body_entered(body):
	if body.name == "Player":
		$AnimationPlayer.play("bounce")
		$Timer.start()


func _on_Timer_timeout():
	emit_signal("coinCollected")
	queue_free()
```

Button.gd contains the script to change to main scene when pressed:

```
extends Button


func _on_Button_pressed():
	get_tree().change_scene("res://Assets/Main_Project_Scenes/Main.tscn")
```

winButton.gd is the script for the button to return to the main menu and releases control of the mouse. This was used for both the You Win and Game Over buttons:

```
extends Button


func _ready():
	Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)


func _on_Button_pressed():
	get_tree().change_scene("res://Assets/Main_Project_Scenes/Main.tscn")
```

References:

- The Godot Game Engine: https://godotengine.org/
- Blender: https://www.blender.org/
- Walking Professors are based on "Gilbert" by Binngi from OpenGameArt.Org and used under a Creative Commons license
- Menu Menu Music is CakeTown by Matthew Pablo from OpenGameArt.org and used under a Creative Commons license
- Game Music is Happy Arcade Tune by Rezoner from OpenGameArt.org and used under a Creative Commons license
- Win Music is Win Music #3 by Remaxim from OpenGameArt.org and used under a Creative Commons license
- Lose Music is Total Fail by Congusbongus from OpenGameArt.org and used under a Public Domain license
- Thanks to BornCG, GDQuest, and Jeremy Bullock for their informative tutorials on GoDot.