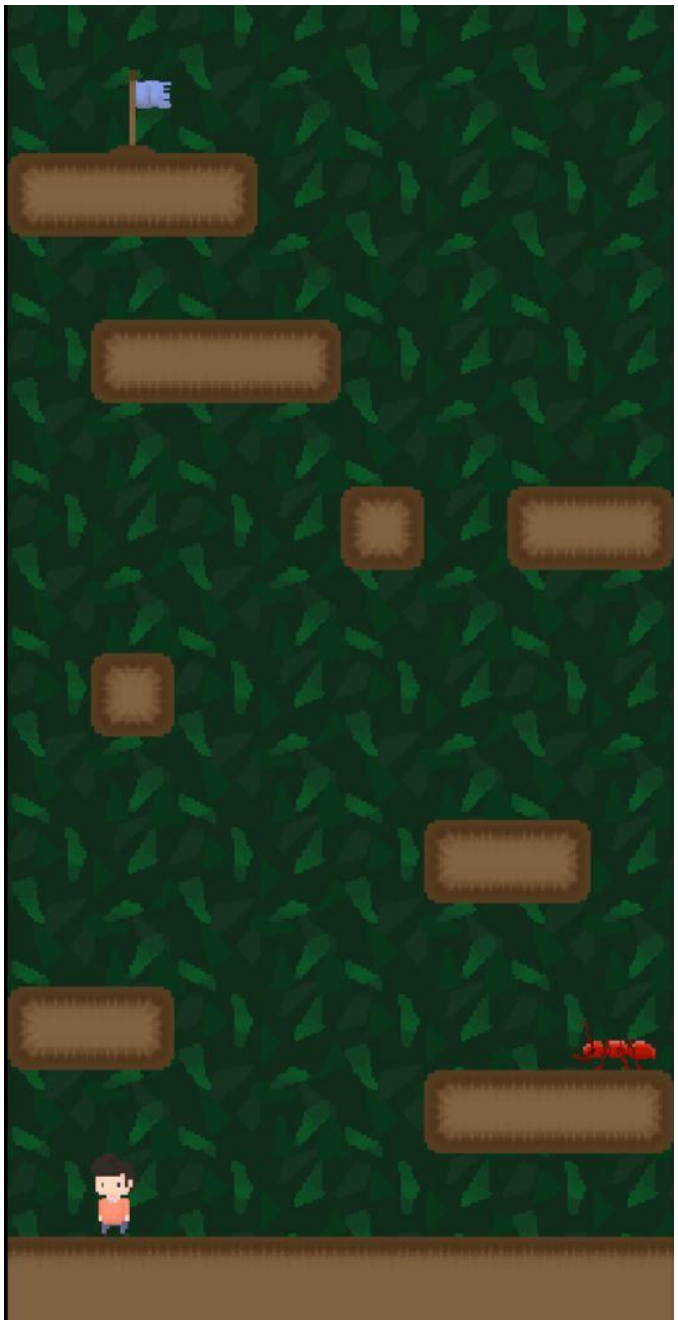
Introduction

Simon’s Climb



Getting Started:

1. Make sure to have python 3.10.8 installed
2. Have pygame 2.1.2 installed
3. Go to this repo: **https://github.com/samuelaa02/Simon-s-Climb**
4. Download the source code
5. Run “Game.py”

Game Controls:

1. Use left and right arrow keys to move and to influence velocity
2. Up arrow key to jump

Game Design

Mechanics/Technology

- The gameplay loop involves the player jumping from platform to platform, avoiding enemies along the way and reaching the flag at the top of each level.
- Core mechanics include: Jumping, moving, avoiding enemies and collecting eggs
- The gimmick is that enemies will push the player rather than kill them and that the different enemies push the player in different ways.
- The reason our game is different is that the enemies can push the player and essentially make them restart the level from the bottom.
- Our game utilizes the strengths of the engine by being in 2-d, and using a lot of rects to handle collisions and movement.

Story

- Simon’s dog was birdknapped by a predatory bird and his goal is to rescue him by climbing to the top of the tree.
- Simon, an everyday guy who loves his dog, is motivated by the birdknapping.
- The predatory bird is the main protagonist and the bugs are his henchmen. The goal of the bird is to try and get their eggs back, so in retaliation, he took Simon’s dog

Player Experience

- Emotions we want our players to experience include but are not limited to: anger, frustration, fear of falling, and satisfaction
- Challenges include, frame perfect jumps, pushy bugs that hinder progress and their own ego.
- Rewards for our game mostly involve self satisfaction, and 100% completion by collecting all the eggs.
- Feedback for the player includes: responsive controls, alternate ending after all the eggs are collected and being pushed by a bug when it touches them.
- The tree backgrounds help to set the scene and the hand crafted sprites help give the player an idea of what they are interacting with.

Game Design Changes

- The original design included health, animations, obstacles in the environment such as sap that would slow the player down, different pickups such as health and jumping buffs, and a dynamic camera that would follow the player. The enemies also were originally planned to have health and to deal damage to the player rather than push them. We also had a final boss battle planned with a health bar but we didn't have the time to implement it. In the game's current state, all of those features were cut out and we decided to not include health for entities and to not have the sap. Most of these decisions came down to both of us not having enough time to implement the full envisionment of the game and dealing with other classes and priorities. The biggest challenge was time. We had the capabilities to make our envisioned game, but just not the time.

Game Development and Documentation

Codebase Outline
Game.py - initializes the model, view, controller, and controls the framerate.
In a while loop, updates the controller and view.

Controller.py - handles broad control over events that occur in the game.

Class Name	Function Name	Function Purpose
Controller	handle_events(self)	Gets the player input and affects game accordingly
	update(self)	Updates the model for the frame (physics mostly)

Model.py – holds all information regarding the games current state

Class Name	Function Name	Function Purpose
Model	changeStage(self, nextStage)	Loads the next stage from the file path arg given
Level		Holds information pertaining to the loaded level

Stage.py – Handles all of the information that is needed to make a stage

Class Name	Function Name	Function Purpose
Stage	load_level(self)	Parses text from a file to gather information to create a

		stage
Platform		Holds information for a platform in the tile grid (material, location)

Entity.py – contains functions and data structures to handle dynamic objects in the level

Class Name	Function Name	Function Purpose
Entity	physicsUpdate(self, model)	Calls functions to check collisions, update positions, and update velocity of entities
	getCollisions(self, model)	Scans the model to get current collisions with any platforms
Player(Entity)	playerJump(self)	Applies velocity to the player upwards and sets the grounded variable to false
Enemy(Entity)	enemyUpdate(self,model)	Calls functions to perform ai logic, physicsUpdate, and logic for player collision
	knockbackPlayer(self,Player)	If the player collides with the enemy, perform appropriate logic based on enemy type
	aiMovement(self,model)	Perform appropriate behavior based on current state and enemy type
Collectible	flagAction(self, player)	Perform specified action when colliding with a flag object
	eggAction(self, player)	Performs specified action when colliding with an egg object
	collectibleCollide(self,entity)	Check if the entity is colliding with the collectible and perform appropriate action if true

View.py – Handles updating the display based on the information stored in the model

Class Name	Function Name	Function Purpose
View	updateResolution(self, new_res)	Updates the pygame display to the new resolution arg
	updateView(self,model)	Draws all of the model information to the screen, updating the resolution if needed
	drawBackground(self,stage)	Draws the stage background image
	drawStage(self,stage)	Draws the platforms in the stage
	drawEntities(self,entities)	Draws the entities in the stage
	getPlatformVariant(self,material)	Returns the appropriate image for the specified material, or a missing texture if the material is not valid

Bugs and Flaws

- When fighting the final bird boss, the player can get to max vertical velocity by jumping under the bird while it descends.
- There are some large bugs in this game such as beetles and ants.

Collaboration

- Version management was facilitated by Github.
- Communication was facilitated between members through Discord.

Group Member Roles, Tasks, and Performance

Sam	Luke
- Character Sprites - Level Backgrounds - Level Design - Entity Collisions - View	- Class Structure - Data organization - AI - Game Doc

Timeline

March 14, 2023	Image file Structure and Sprites
March 27, 2023	Basic File Structure
March 28, 2023	Implemented File Parsing for Level Data
March 30, 2023	Basic Functions for MVC were implemented
Milestone 1	
April 3, 2023	Completion of File Parsing for Level Data
April 18, 2023	Implemented Level 1
Milestone 2	
April 20, 2023	Added Player Controls and Entities
April 24, 2023	Added Collisions and Collectible Sprites
April 25, 2023	Added Enemy AI
May 1, 2023	Added Level 2 and 3, Enemy Collisions, Collectibles, and Increased FPS
Final Game Submission	

Links

Github Repo –
https://github.com/samuelaa02/Simon-s-Climb

Demo Video –
https://drive.google.com/drive/folders/1cgA3pfKBfeQitxnHhFgT80qVlrV3RIA2?usp=share_link