Project Documentation

Dakshesh Mishra

Contents

1 Modules

The modules used in the project are:

1.1 pygame

1. Used to handle the game window, rendering and screen updates
2. Loading and displaying the images
3. Capturing user inputs such as mouse clicks and keyboard events
4. Managing frame rate

1.2 random

1. Used to ensure that the game is not unfair for any of the players by randomizing the player to move first and the bird generation

1.3 sys

1. Exiting the game cleanly

1.4 numpy

1. 2-D vectors to represent body locations on the screen, velocities and accelerations.
2. Collision detection and implementing physics properties such as friction and rotation.
3. To carry out large calculations efficiently and accurately

1.5 os

1. Managing file paths for images
2. Checking for and creating leaderboard save files
3. Ensuring compatibility across systems
4. Dynamically loading levels

1.6 re

1. Regex matching to detect game-level files and leaderboard files

2 Directory structure

Figure 1: Directory Structure

birdpowers.py Functions that power up birds

body.py Definition of the body class that is used to implement physics

builder.py Functions that are used to read level-log files and create environments from them

collision.py Used for collision detection and finding points of contact

contact.py Class for objects that store the contact points and other related properties for every two bodies that are in contact

damage.py Function used to calculate damage that is dealt upon collision with a projectile to the structure

draw.py Functions to draw various in-game elements such as buttons and text boxes

environment.py Container for all bodies where all collisions are resolved using relevant physics principles.

leaderboard.py Used to store and process the leaderboard for each level

Modules Contains all user-defined libraries

levels Stores level log files

leaderboard Stores leaderboard log files

images Stores images that are required

3 Running Instructions

1. The game is started by running the python script ‘main.py‘ in the parent directory.
2. The game boots into the main menu which shows 2 options:
   (a) Select Level:
      i. Allows you to look at the available levels that can be played
      ii. Upon choosing one of the available levels you are prompted to enter the names of the first and second player.
      iii. After entering the names, the final game screen opens and you can play the game.
   (b) Leaderboards:
      i. Allows you to look at the available leaderboards, which contain the high scores of each level that has been played previously (in the file that is submitted some sample data are included in the leaderboard already to show its functionality)

4 Game Mechanics

4.1 Types of structures

1. Wood
2. Ice
3. Stone

4.2 Types of birds

1. Red
2. Chuck
3. Blue
4. Black

These birds have special properties regarding the damage they do:

1. Red - These do balanced damage to all types of structures.
2. Chuck - Does more damage to wooden blocks and slightly lesser to others
3. Blue - Does more damage to ice blocks and slightly lesser to the others
4. Black - Does more damage to the stone blocks and slightly lesser to the rest

1. The game creates two environments, one for each player. The environment for player 1 consists of the structures on player 2’s side and the attacking bird of player 1. Similarly the environment for player 2 consists of the structures on player 1’s side and the attacking bird of player 2.
2. The game proceeds in a turn-wise manner where the players take turns launching birds at the other player’s structure, the first to destroy the other person’s structure wins.

4.3 Details of the mechanics

1. The first player to move is decided by randomly choosing a number among 1 and 2.
2. Bird Generation:

   A bird is chosen at random with equal probability from a dictionary containing all 4 of the birds at the start of each round of launches.
3. Scoring system:

(a) The score of player i after they have made n launches is given by the formula:

$$\frac{\sum \text{Health of block destroyed}}{n}$$

4. Structure Destruction:

   (a) Structures can be destroyed in one of two ways:

      i. By dealing damage that is greater than their max health points

      ii. By pushing them out of the playing area

5. Turn ending:

   (a) The turn of a particular player ends in one of two ways:

      i. It has been 10 seconds since their bird was launched

      ii. To end the turn sooner the following quantity is calculated, if $Q < 0.01$ the motion of all blocks has essentially ceased and the players turn is ended.

$$\sum ||\text{velocity of block}|| \leq \text{minV}$$

6. The game is won by the player who manages to destroy all of the opponents' structures before their structures are destroyed

7. Leaderboard inclusion:

   (a) The winning players score and name is included in the high score leaderboard that is maintained across runs of the game.

   (b) After the game is finished if the player wishes to go back to the main menu they can click on the top left corner of the window to go back to the main menu

5 Features implemented

5.1 1. Basic Features

5.1.1 a. Game Interface

1. A UI was developed with main menu and the ability to enter player names before a game begins.
2. The player also has options to move back and forth between the various game scenes

5.1.2 b. 2-Player Gameplay

1. The game has been designed so that the players can throw projectiles at each other in a turn by turn manner (first mover being randomly decided)

5.1.3 c. Projectile Mechanics

1. A gravity based launching system where the projectile undergo proper parabolic motion because of the gravitational acceleration has been implemented.
2. The birds can be dragged using the mouse and released by releasing the left mouse button.
3. The farther the bird is dragged from its origin the faster it is launched.
4. A limit has been enforced in the amount the catapult can be stretched so that the birds do not fly too quickly to be seen and to prevent tunneling.

5.1.4 d. Game over conditions

1. The game ends when one of the players has entirely destroyed all of their opponent's structures.
2. Each structure can be destroyed by either damaging them or pushing them outside of the game window.

5.1.5 e. Projectile effects

1. Stretching the catapult while launching the projectile shows the approximate trajectory of the projectile using white dots.
2. A power up has been added so that while a projectile is flying, left clicking the mouse speeds it up.

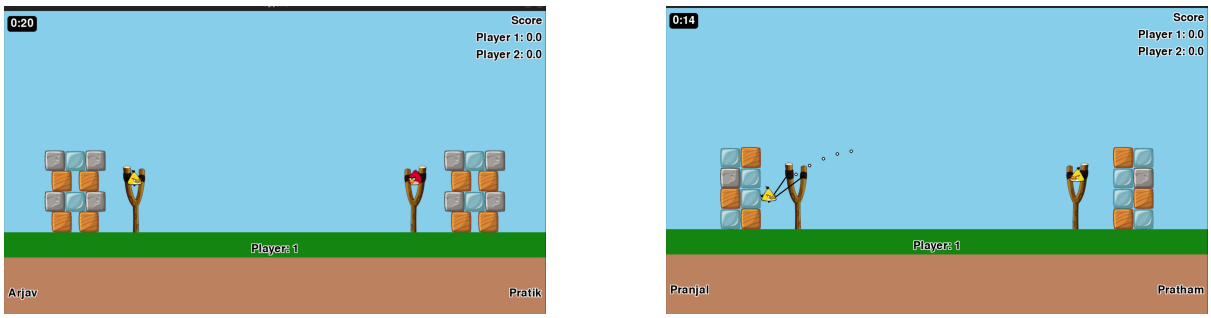

Figure 2: Basic Features

5.2 2. Advanced Features

5.2.1 a. Game Levels

1. 3 Levels have been designed in order of increasing difficulty for the player to enjoy.

5.2.2 b. Log Files for levels

1. To make it easier to add levels into the game a log-file like structure has been implemented.
2. The player can create two text files in the levels directory "level{xyz .txt" and "level{xyz}p2.txt", where each line in the former stores information of the blocks that have to be added to the environment of player 1, and the latter stores the blocks that have to be added to the environment of player 2. The game automatically recognizes these files and a new level will show up in the "Select Level" menu.

5.2.3 c. Scoring system

1. A scoring system is devised so that instead of just having a winner and loser every game we can have a leaderboard which compares the performance between two different matches as well
2. The score of particular player at the end of n turns is given by: (The formula is missing in the provided text)

5.2.4 d. High Score Leaderboard

1. A log-file like structure is used to store the performance of the winner of a particular match in a leaderboard, which can track performance across sessions.
2. Each level has its own leaderboard.

5.2.5 e. Destructible Structures

1. Depending on the health remaining in a particular structure the sprite that loads for it changes.
2. There are different sprites for each of the structures.
3. The first sprite is displayed when the health of that block is greater than 75%
4. The second sprite is displayed when the health of that block is less than or equal to 75% but greater than 50%
5. The third sprite is displayed when the health of that block is less than or equal to 50% but greater than 25%
6. The last sprite is displayed when the health of that block is less than or equal to 25%

5.2.6 f. A Physics Engine

1. A proper 2D physics engine is implemented to show realistic collisions, rotation, and friction.
2. Collision detection is done using the Separating Axis Theorem (SAT).
3. The colliding points are detected and used to apply rotational impulses as well as frictional impulses.
4. This gives the game a much more realistic feel.

6 Project Journey

6.1 1. Developing the physics engine

I wanted to simulate realistic collisions, but did not know how to implement them in python. I found an article in the GameDeveloper's Magazine that detailed how collisions can be implemented in games to make the game realistic. Along with this I saw the tutorials and codes of a few people who had implemented such a physics engine from scratch to get inspiration for my own code.

6.2 2. New to pygame-ce

Pygame-ce is a library which I have never used before. Naturally, there were a lot of hurdles from understanding how to open a basic window using pygame to how the pygame.image.rotate() function worked. To better understand what the functions were doing I went through the official pygame-ce documentation and tutorial codes on their website as well as a few youtube tutorials.

6.3 3. Deciding values for physical constants

A lot of experimentation was required to arrive at values of physical constants such as the coefficient of restitution and the coefficient of friction as well as multipliers that were to be applied during the launch, etc. Experimenting through different values I was able to find a set of constants that made the game seemed realistic to me.

6.4 Learnings

The main learning through the project journey were:

1. How actual games detect collisions using the separating axis theorem, how these collisions are resolved both translationally and rotationally.
2. The importance of having different files for different functions as this made the development of the physics engine so much easier. I was able to treat objects as black boxes making it simpler to implement the physics engine as well as develop the game.

7 Bibliography

References

[1] Pygame Documentation. https://www.pygame.org/docs/

[2] simple-2d-constraint-solver https://github.com/ange-yaghi/simple-2d-constraint-solver

[3] A Primer on Pygame. https://realpython.com/pygame-a-primer/

[4] Chris Hecker. Rigid Body Dynamics Part 3 https://www.chrishecker.com/images/e/e7/Gdmphys3.pdf

[5] Chris Hecker. Rigid Body Dynamics Part 1 https://www.chrishecker.com/images/d/df/Gdmphys1.pdf

[6] Chris Hecker. Rigid Body Dynamics Part 2 https://www.chrishecker.com/images/c/c2/Gdmphys2.pdf

[7] Chris Hecker. Rigid Body Dynamics Part 4 https://www.chrishecker.com/images/b/bb/Gdmphys4.pdf

[8] Toptal. Video Game Physics Part II: Collision Detection for Solid Objects. https://www.toptal.com/game/video-game-physics-part-ii-collision-detection-for-solid-objects

[9] Erin Catto. box2d. https://github.com/erincatto/box2d

[10] Invent with Python. Pygame. https://inventwithpython.com/pygame/