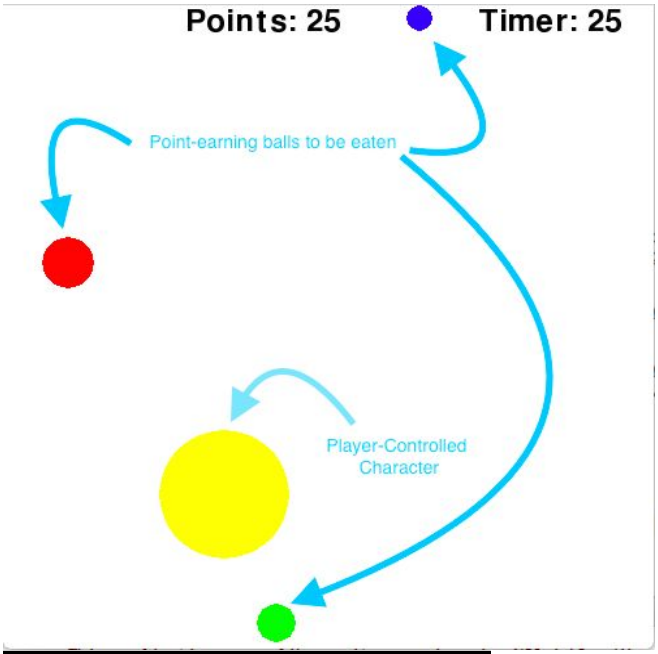
Rebecca Gettys and Liv Kelley

Interactive Programming Project

Project Overview

Our project is a simple arrow-key controlled game based on Pacman. The object is to gain the most points in the shortest amount of time by aligning the main character (a yellow ball) with the ball with the highest point values, which correlates with the size and color of the ball. As soon as a ball has been "eaten" it moves randomly away so the chase can continue until the time runs out. From this project, our team has learned how to utilize functions of pygame and how to better use objects, fulfilling the learning goals that we set at the start of the projec.

Results



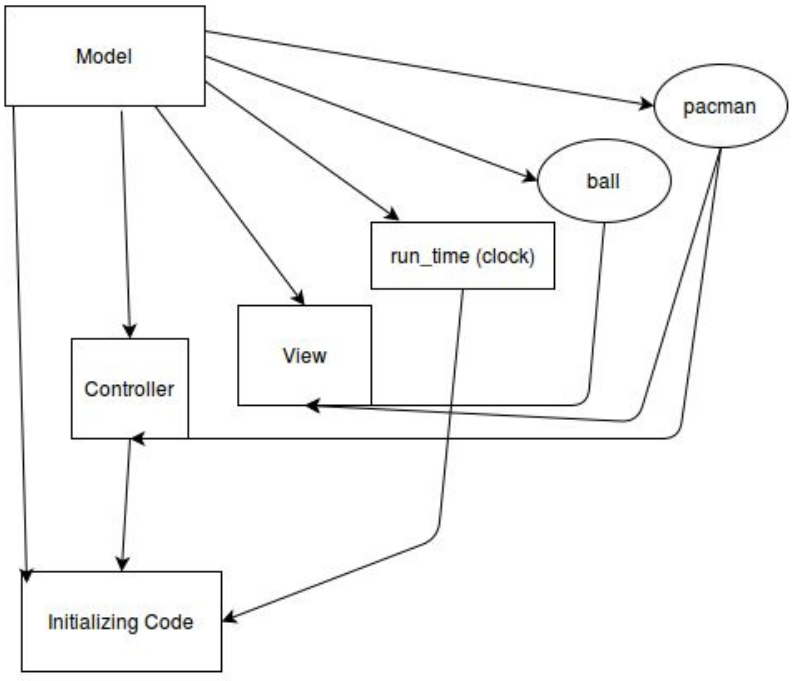
The team created a pacman or snake-like game called "Polka Dots" in which the player controls the character (a yellow ball) with the arrow keys. When the player runs into a "point" ball (blue - 5 points , green - 10 points, or red - 15 points) and centers the yellow character ball underneath it, the character "eats" it, earning different numbers of points for different balls (figure 1). The screen also displays the time remaining and the points earned. The ball that has been "eaten" then randomly moves to a new location on the screen and can be eaten again. This continues and the player tries to eat as many balls as possible during the time period. When the time period (60 seconds) is up, the final number of points that the player earned during the game is displayed (figure 2)! In short, it is a race to eat as many "point" balls as possible. The player can press an arrow key once to move 10 pixels in the desired or hold it down to move continuously in that direction; you cannot drive the character off of the screen. The game defaults to a small size window but can be changed within the code without changing other aspects of the game.

Implementation

Our implementation consists of classes named model, “pygame view”, and the controller as well as the lines used to run the code. The model and view classes define what the user sees and interacts with. The objects define each of the elements in the game, like “pacman,” the three balls, and the clock. The model also defines self.running, which determines whether the window is open. The definitions of ball and pacman define how each move and how they update each round and the definition for the clock utilizes pygame’s get_tick to count down in seconds. The running part within the code starts the code running and allows the code to be shut down when the game stops. Pygame view displays the characters, screen, the captions on the screen for “Points:” and “Time:”, the score, and the end screen.

While these two classes define what the user sees, the portion of the code dedicated to starting the program and running the program consists of the controller and the initializing text. The controller defines what happens when a key is pressed based on the x and y axis within the screen. This connects to the initializing text what calls these things into existence. The code for running the program first initializes pygame, the clock, get_ticks, and then key.set_repeats, which allows the controller to work. Next, this initialization code uses inputs to define the dimensions and points associated with the aforementioned “pacman” yellow dot and balls. Then the model, view and controller are explicitly defined and the game is defined as running while True. Together these parts of the code allow the game to run properly and help to define the actions in the game.

Together the two parts of the code can be visualized with this UML diagram:



Reflection

Overall, a lot of this project went very well. In general, the team was very effective at knowing when to ask for help and getting it. One problem the team encountered was having to redefine the project upon discovering the original goal was outsized. The original goal was to create openCV controlled pacman, which didn't turn out to be possible (as Oliver politely warned us.) Despite this, the team responded effectively by scaling back to a game feasible in the amount of time given. The team members realized the extreme scope of the project very soon after starting which allowed for a chance to change course; in the future, both team members will strive to be realistic about a project's scope. Throughout the project the team utilized examples, documentation, and NINJA and instructor help to effectively advance the game without spending too much time trying to code things that wouldn't work. The result is code that is clean, clear, and well-commented, and that the team members understand. The final code also incorporates outside code snippets and resources as needed without becoming 'cobbled code' - many pieces of outside code we used were skeletons needed for us to understand a structure that were heavily modified. Since this was a game, unit testing was very easy, and the game ultimately behaved as the team expected.

Despite the team's success, some things could have gone more smoothly. The team met as planned, divided things up, worked independently and then met to integrate. During time apart Becca tended to get more done than Liv, and this could be very stressful for Liv as she felt that she wasn't going to have a chance to do her half of the project. Becca tried to adjust by intentionally slowing her work on the project down to give Liv a chance to catch up, and this caused Liv to feel rushed. In retrospect, more pair programming would have helped to help reduce this disparity, though the team's busy schedules may have prevented that. In the future, if both partners didn't want to do, pair programming for a majority of the work, one should make sure that in addition to being well matched for skill level, one should also be well matched for work speed and available work time.

Resources and Code Snippets

In the course of this project we used a few resources and code snippets to help. These are listed below.

Starter Code/Softdez Website/SoftDez Floobits code - We used the code and explanations from the softdez website's page on this mini project. Also, we heavily utilized the floobits brickbreaker code as a model for our general structure and used it's initialization, after some significant modification, to get things running (after our original attempt had been badly flawed). In short, we used the floobits brickbreaker to understand how a pygame game "should" work.

Basic pygame tutorial from pygame.org(http://www.pygame.org/docs/tut/tom/games2.html) - We used this to figure out how to create text and blit it to the screen for the point counter.

Event Handling (Keyboard Input) in Pygame - http://www.nerdparadise.com/tech/python/pygame/basics/part6/ and

https://sivasantosh.wordpress.com/2012/07/18/keyboard-event-handling-pygame/. We dabbled in and modified the keyboard control code from Nerd Paradise until we were able to control the game using the arrow keys. Later, when we were trying to make continuous-press motion function (so if you hold the arrow key, the character keeps moving), I stumbled across the Sivsantosh article, which showed me how to do that using set_repeat.

ThinkPython, Python documentation (python.org), and pygame documentation (http://www.pygame.org/docs/) I think this is self-explanatory - we read these all the time when I am not sure of things and used the pygame documentation extensively in debugging, figuring out correct function calls, graphics generation, and generally making our game work.

Inspiration from the timer came from the following youtube videos. https://www.youtube.com/watch?v=xBN2LJA0Src and https://www.youtube.com/watch?v=8d2cWG5J4lc. The former video was used for more specific implementation of pygame.time.get_ticks and pygame.time and the latter was used for the structure of the timer, which allowed it to count-down.

The code we used for creating text within the document came from this source: http://www.pygame.org/docs/tut/tom/games2.html This source gave us a framework for printing text, the points, and the timer on our screen within the view (pygameview) class.