C200 Fall 2017 Project

12/13/2017

Gary Baker, Isaiah Sherfick

Introduction

Breakout is one of the first commercially successful widely-known arcade video games. Originally released by Atari in 1976, it is cited as one of the main inspirations for the ubiquitously popular in-home console game *Pong.* The game is gracefully simple; the player controls a small paddle on a horizontal plane and has to bounce a ball back and forth between the paddle and some bricks at the top of the screen. If the ball goes past the paddle, the game is over. Today, that seems incredibly lacking in terms of features, but in its day, *Breakout* was perfect. It didn't try to do anything it couldn't; it found a simple, addicting formula and executed it well. For our project, we were tasked with more than just recreating the classic, but adding new and exciting features to revamp the game. We added custom levels, a scoring system, and new rules by which to play the game. These tools were meant to challenge what we had learned throughout the course, and to help broaden the knowledge and critical thinking capabilities that we had constantly been improving.

Team

Our team, which originally consisted of three members, is unfortunately turning in this project with two names on it. Gary and Isaiah communicated and contributed heavily to the project (since they are roommates, this was very easy to do). Kira, however, failed to respond to our reaching out to her, and she contributed nothing. At the start of the project, we divided the basic elements of the game between the two of us (the paddle, the ball, the bricks, etc.), and we wrote separate class files for each of these items. Later on, as the code became more complex and most of the work lied in the main file, the work became more of a paired programming

exercise. One of us would sit at Gary's computer and write code while the other sat nearby, bouncing ideas off each other.

Gary's strengths lied in his ability to make really graceful solutions to problems that could easily become a huge complication. For example: he was the one to implement the game reading a string to place the bricks instead of instantiating a new brick for every single brick in the game. He also did all of the external graphics for the game, and those are one of the biggest strengths of our game. He also was the graphic designer, and created different types of bricks and the screens between levels. Isaiah's strengths were his ability to quickly write code that worked. When he had to get something done, he just sat down and got it to work; we could make it look nice later. Our weaknesses lied on opposite ends of the same spectrum. Anytime a new feature needed to be implemented, Isaiah's immediate response was to simply throw it in the main loop, whereas Gary would suggest a new separate file and a dedicated function. We worked around this by finding a happy medium for most of the features.

Features Implemented

We had to implement many features throughout our time developing our version of breakout. We implemented simple tasks, such as taking user input and converting that to the movement of the paddle, random initial direction for the ball, or drawing the bricks to the screen, and more complex tasks such as unit collision, changing levels, and ball side detection. Every new feature had a hurdle to overcome, whether it be a bug, a typo, or incomplete logic , and these created some monstrous headaches along the way, but they helped smooth out and improve many features of the game. Some of the features included are:

- Feature: Paddle Movement
- Feature: Creation of Bricks as Objects
- Feature: Brick Destruction
- Feature: Scoring System
- Feature: Ball Collision
- Feature: Random Initial Ball Direction
- Feature: Changing Ball Direction Based on Paddle Collision
- Feature: Additional Levels
- New Feature: Animated Start Screen

- New Feature: Easy to Use Level Designer Based on Strings
- New Feature: Constant Ball Speed Based On Pythagorean Theorem
- New Feature: Falling Bricks Based on Level
- New Feature: Bricks With Only One Breakable Side
- New Feature: Pause

Beyond the Standard Features

Going beyond the standard features was a rough start. First, we had to decide what we believed was doable in the time span that we were given, and this was made especially hard by us having to find a good time to not only talk about what we were going to add, but a time to actually implement it. Since the members of group eleven are roommates, and also had conflicting schedules, this often meant working late in the evening, when both we unmotivated to be doing any sort of work whatsoever. Despite this, the group made it a clear objective to work on the project almost every night. The commits might reflect otherwise, but there were actually a select number of nights spent standing around a whiteboard as we talked talked with one another about what the next step was, how feasible an idea was, and trying to sell our ideas to each other. After getting through this process, we implemented many new features.

The first to be implemented was the level designer. It is contained within the levels.py document, and consists of two parts. All a person has to do to create a level is write it as a string with certain numbers representing certain bricks. Then the level_load function will convert these strings into a list of bricks that will be drawn to the screen. These bricks were objects that the ball would collide with, and some would have special properties, and all that was stored based upon the level_designer.

The next to be put into the system would be the constant ball speed. This one was actually a doozy at first. We wanted to change the direction of the ball, without changing its speed. We had many horribly failed attempts, such as sending the ball at infinite speed in one direction, having it blit back and forth that it appeared to be in two positions simultaneously, and having it ignore all collision, but by a large amount of brainstorming, we ended up with what we now use to get the ball speed. This is based upon using the Pythagorean Theorem, and it helps make the game as fair as possible between each playthrough.

The third feature we added was the falling of the bricks past the first level. This required changing many of our pre-existing objects, functions, and files. As anyone may guess, this had caused quite a number of crashes, but there was always a way to patch it up, In a shorter time the we believed it would take, we had bricks not only falling, but falling faster at each new level. We had to add an entire new function to the brick class, change the way the bricks were created, and basically do a small overhaul on the load_level function.

The fourth feature we added was bricks that would only break on certain sides. We had already had a function that detected what side the brick was colliding on, but it was extremely fragile and needed a lot of refinement. After doing so, we added another attribute to the brick that would be set to one of five strings. Then, using a combination of the string and the attribute of the brick, it would decide whether to remove the brick from the level or not.

The final new feature we implemented would seem as if it was the simplest, but was easily the most frustrating part of the project to work on. We added an animated start screen. The idea was simple; we created three images, and after a select number of frames, the image would swap to the next one in a forever endless circle. We came to the problem of it always going to the same picture, and then not changing anymore. After hours of debugging, rewriting, and double-checking, we found that the problem was nothing other than the difference between an if and an elif.

Software Development Process

When we boil our software development process down to its simplest parts, we really had only a few ways in which we worked, and we altered them according to what was need at any given time. Our first process was to display our ideas on the whiteboard to one another, and to set smaller objectives and goals for each individual. Towards the end, we worked on goals more together, but the initial idea of dividing the labor after brainstorming was there. We would also use this time to go through and explain what the code we wrote actually did and how it worked, so that the other would have a great sense of how the entire game worked, and what the best way to implement new functions and features would be.

The second way in which we worked was to have each of us at our respective computers working on a different part of the game. This was commonly used in the beginning, but as we came closer to the end, and as we had to use more overlapping files, we tended to sit

around one computer discussing the best way to go about the task at hand, and then having both hands work on the same computer. This proved to be the best way for us to implement extended solutions, and to solve difficult problems. Large scale endeavors were handled with both of us being able to think things through.

In terms of what we learned that we didn't expect to, we had no idea Github could be such a headache. Throughout the rest of the semester, we had basically just used Github as a personal cloud-based flash drive, and we had never had to deal with syncing conflicts, branching, or any of that stuff before. We actually learned how to make it work pretty smoothly, and we grew to appreciate its power in the development process. We also learned, quite humorously, that if two python files attempt to import one another, bad things happen.

In the process, Google was a great help when it came to our Github aches. Basically, that's all we needed to iron out all the problems we were having.

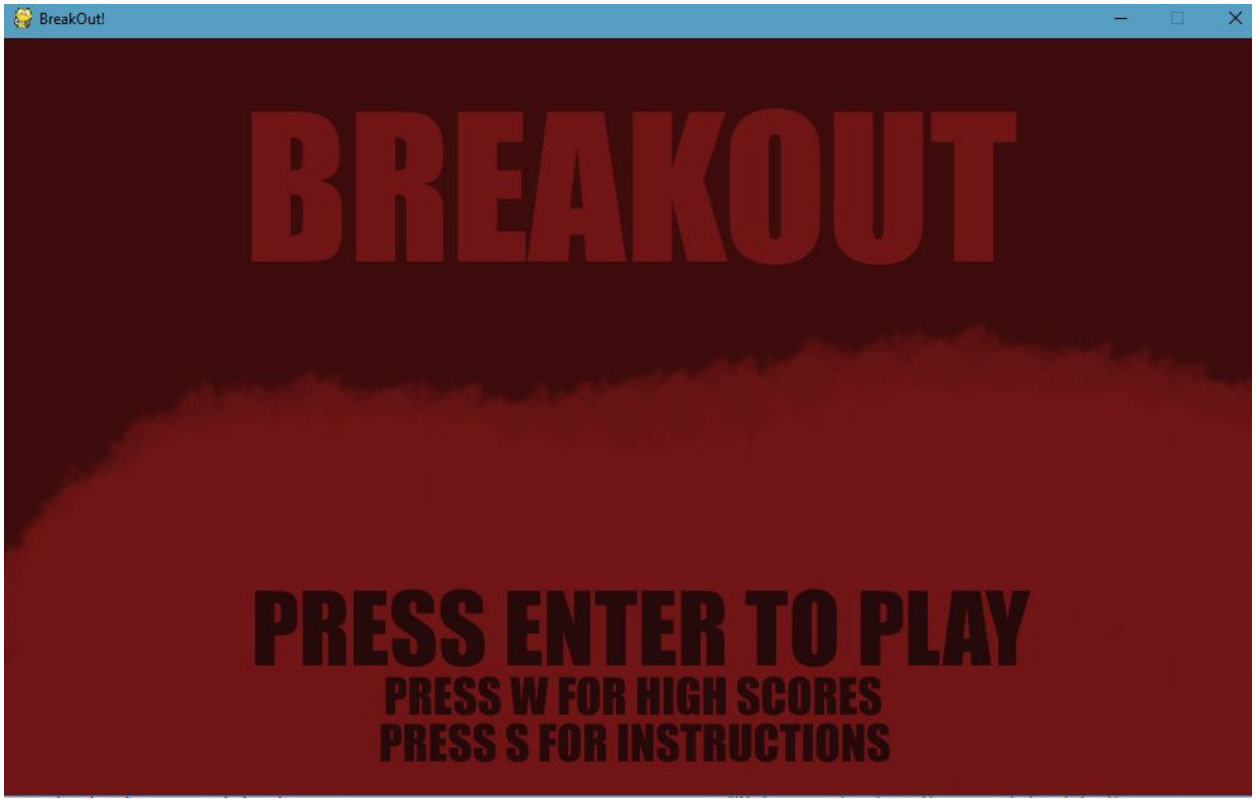
There are a few challenging aspects of working as a team that are neck and neck for the worst position. One of them is trying to communicate ideas to one another while they're still being formed in your head. When you've both been staring at a problem for hours with dreary eyes and tired minds, it's hard to put ideas for code into intelligible sentences. Adding onto that, another problem we encountered was convincing one another that our solution *would* work were we to implement it, as we didn't want to waste time writing code that wouldn't work. These two issues fed off one another very heavily. That being said, we believe that these skills just come with experience in the field, and we don't really have any suggestions for improving the development process.

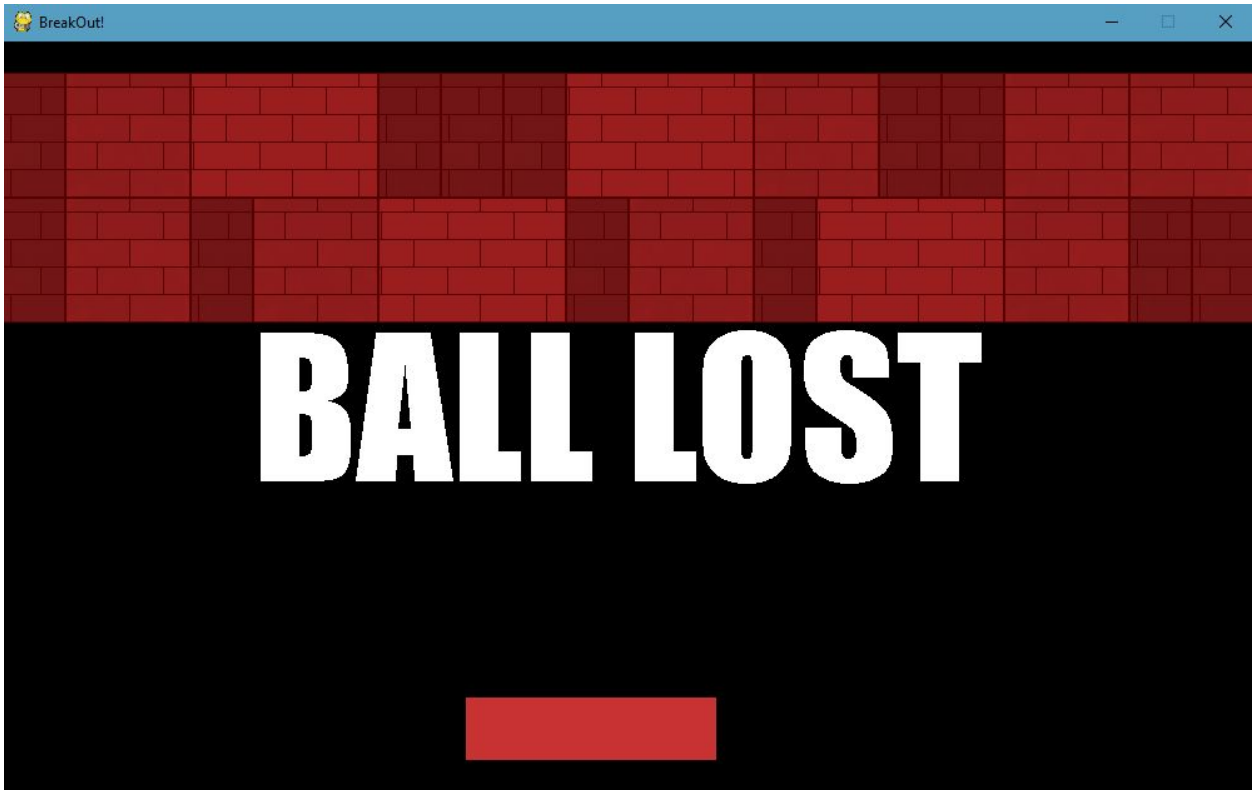
Future Work

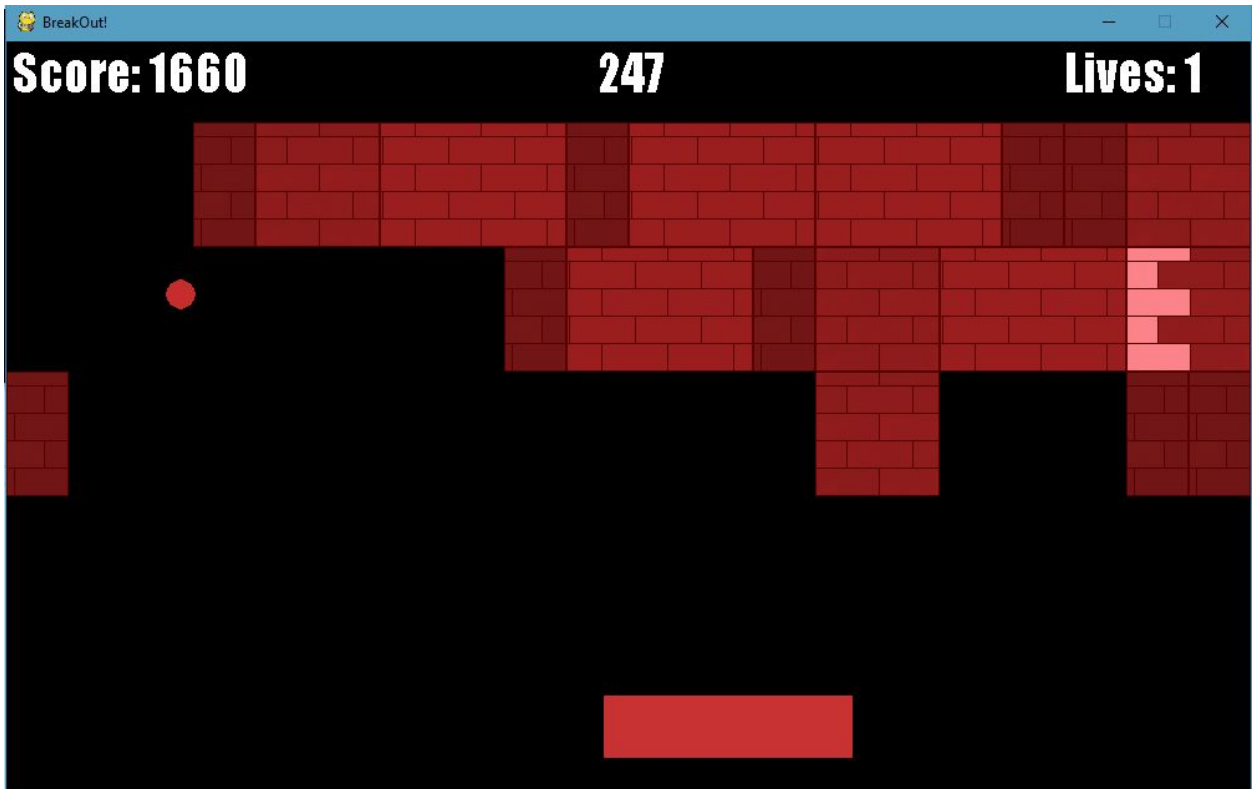
If we had more time to work on this, there are a few features that we really wanted to fit into our game. First and foremost, we would have made the ball's angle influenceable by the player depending on where the ball collides with the paddle. As it is, every time the ball "bounces" off of something, one of the direction values is simply multiplied by negative one. If we had more time and were more brushed up on our calculus, we would have made it so that the closer the ball was to the edge of the paddle, the harsher the angle it bounced at would be. We also wanted to add a lot more levels with different themes and more challenging unique properties, such as certain bricks granting the player power-ups or being bombs. In addition, we would

have liked to add a lot of sound effects to the game, giving it more of an impact for the player when the ball hit something. It also would have been a really interesting project to try to make the game have a high score database stored on the internet that it updated every time it played. Finally, the current version of the game has slightly buggy collision detection, and if we had more time we would add interpolation to it to fix that.

Images




BreakOut!
BALL LOST


BreakOut!
Score: 1660
247
Lives: 1

Code Listing

- MainLoop.py
- ball.py
- bricks.py
- graphicsload.py
- levels.py
- paddle.py
- Savescore.py
- Pygame
- Sys
- Math
- Time