PolyGolf Report

160003289

December 2019

Contents

1 Design

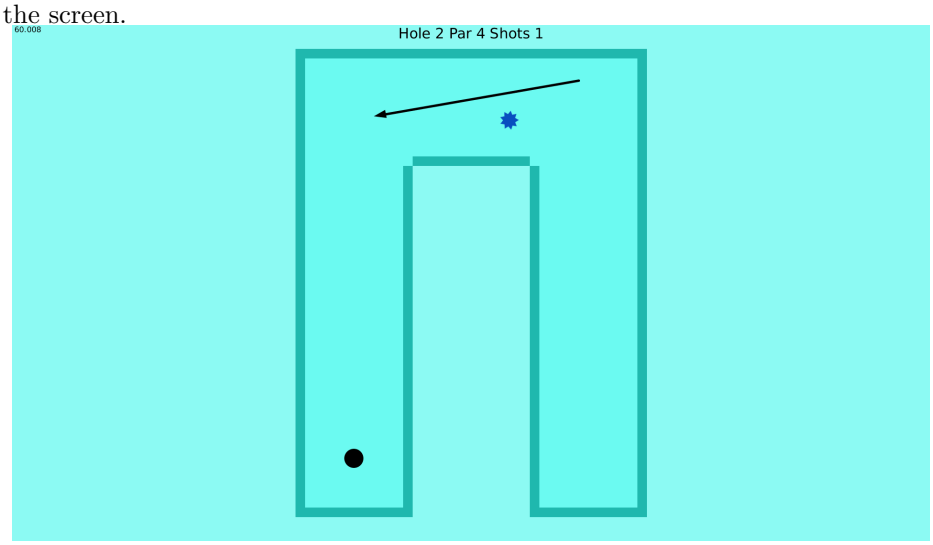
1.1 Overview

For my final game, I made PolyGolf. This is a 2D mini golf game with the twist that you are playing with a polygon rather than a circle, hence the name. The game relies on polygon to polygon collision to generate collisions that result in

changes to both angular and linear velocity. Depending on which polygon you are using, this can cause slightly unpredictable collisions depending on their different properties and add more fun to the game.

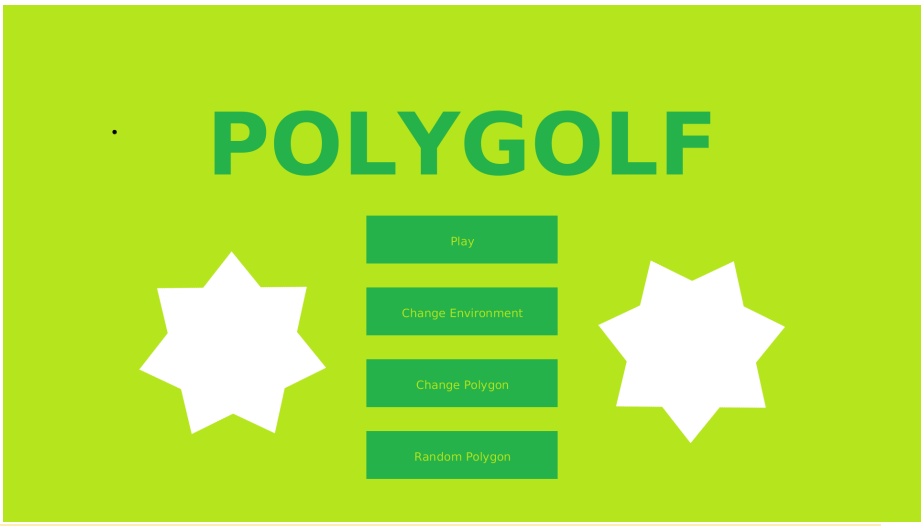
1.2 Game

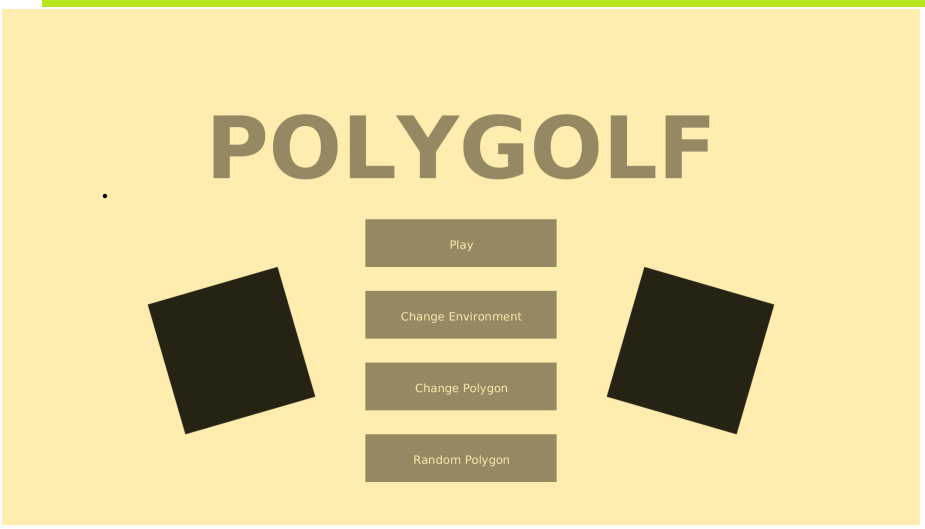
In PolyGolf, I have kept the gameplay simple by making shot taking as easy as drawing an arrow which is applied as a force to the polygon. The assumption is that most users will understand the basic concepts that appear in all golf games such as each hole having a par and only being able to hit the polygon while it is stationary. I designed the cup so that the polygon will only drop if its straight at the hole or not going to quick at the edges. I did this by setting a speed cap for a shot down the middle going in and then tailing this off linearly to the edges. Here is a screenshot of a shot underway with hole information at the top of the screen.



1.2.1 Menu

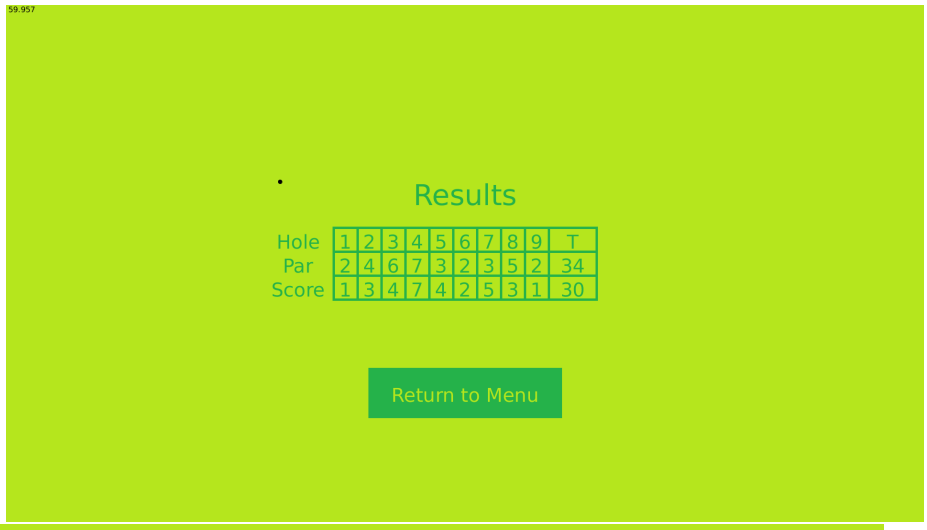
The menu has four simple buttons allowing you to play, change the environment, change the polygon or choose a random polygon. The currently chosen polygon is displayed on screen as well. The menu is shown below with two of the different environments and two different polygons.



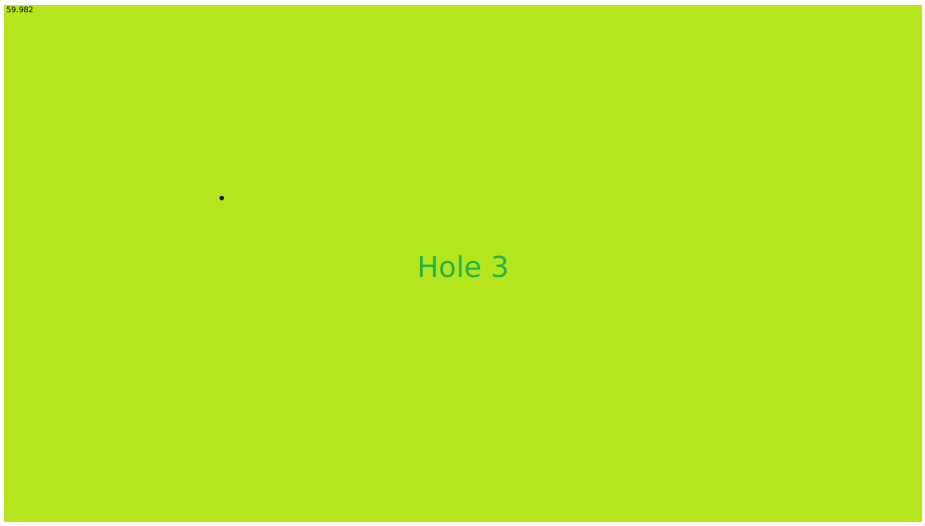


1.2.2 Results and Scores

I display the score for the previous hole and the hole number for the next hole between holes and during the hole, information about the hole is shown at the top of the screen. At the end of the game, a results table is displayed in the standard golf format, with the hole numbers, par and score listed, with the totals at the end. A results table and the two between hole screens are shown below.


59.957
Results
Hole 1 2 3 4 5 6 7 8 9 T
Par 2 4 6 7 3 2 3 5 2 34
Score 1 3 4 7 4 2 5 3 1 30
Return to Menu


0.087
Bogey



1.3 Mechanics

1.3.1 Polygon generation

For this game I had to make methods to generate different polygons for the player to use. I started with a simple method from the processing website for generating a regular polygon which gave me a good set of basic shapes. I then decided to add stars to my game and using the method described at `http://mathworld.wolfram.com/StarPolygon.html` to generate polygons by connecting every qth point from a regular polygon. This generalised function also allowed me to generate regular polygons with q = 1.

1.3.2 Hole Design

For hole design in PolyGolf i had originally planned to use procedural generation. However I didn't have time to attempt this and instead defined my own course. Each hole is a collection of tiles arranged in a grid with each tile having a set of specified borders and an obstacle pattern. This allowed me to generate a tile with a specific obstacle pattern but pass in which set of borers I want to be added. This reduced the number of methods I needed ti generate tiles significantly. In my hole generation I defined which tiles I wanted and where and the first and last were automatically defined to the start and end points of the hole. I then had a course generator to fill holes into a course and if id had more time to make holes, I had implemented all the functionality to support multiple courses.

1.4 Physics

The making of this game involved substantial work with physics. The physics involved include drag (simulated for both linear and angular velocity with a damping factor), collision detection, collision resolution and applying forces for both linear and angular acceleration. Originally when starting to make my game, I investigated the particle and rod system shown in lectures due to the way it easily simulates rotations of the polygons. However I couldn't make the polygons generated in this way keep their shape to a satisfying degree and so decided to use a rigid body system of a point mass particle with a collision box around it.

1.4.1 Collision Detection and Contact Generation

For collision detection, the first thing I did was bound my collisions. I did this by adding a bounding circle to every shape as a simple way of reducing the number of collision checks. Firstly I checked which game tiles the players bounding circle could possibly colliding with and then for each of those tiles, I checked if the players bounding circle collided with any of the obstacles on the tile. Once I reached this stage, I started using the collision detection algorithm described in Millington section (13.3.5) for box and box collision. This collision detection describes 6 different case of collisions for boxes but I decided only the point-edge collision was relevant to me which could be run in both directions to get how points from each polygon interacted with the other. Looking back I should have also looked at edge-edge collision as my final game has issues when two separate vertices hit the same obstacle at the same time as it resolves both contacts independently, creating a net gain in impulse from the collision.

Now that I have 2 particles with bounding circles that collide, I check for collisions between each point of each particle and each edge of the other particle. I start by first using the algorithm shown here `http://www.jeffreythompson.org/collision-detection/poly-point.php` to find if the point lies within the polygon and then I check the penetration depth of any colliding points with each edge. I take the minimum penetration distance for the colliding point and generate a contact at the edge that gives this. This contact is added to a global list storing contacts and each contact is told the two particles involved, the point of collision relative to the first particle, the contact normal and the penetration depth.

1.4.2 Contact Resolution

Once I had my contacts generated, I had to process them. This was probably the most challenging piece of physics due to the rotating polygons involved. The first step of resolving each contact was to resolve the interpenetration. Originally I started trying to implement nonlinear projection to provide a realistic result but due to time restraints I decided to first implement a proportional system where each particle is moved by a percentage of the penetration proportional to its mass and I never had time to come back to this.

I then had to resolve the velocity from the contact. For this I used the method described in Millington (14.2) but I kept my coordinate system as the world coordinates for the duration of the collision due to the simpler nature of my 2D system with all rotations about the z-axis. Firstly I calculated the desired change in velocity which is the difference between the separating velocity and the closing velocity by summing the linear velocities for both particles and the linear velocity caused by their angular velocities at the collision point and negating this by a multiple of (1 + Coefficient of Restitution). I then calculate the velocity change per unit of impulse by calculating the sum of the angular and linear velocity changes per unit impulse. The angular velocity change per unit impulse is the amount of impulsive torque generated from a unit of impulse in the direction of the contact normal (cross product) and this is then multiplied by the inverse of the moment of inertia for the particle (discussed below). The cross product of this value with the relative position of the contact position then gives the linear velocity at that point contributed by angular velocity.

Finally I then calculate the impulse required for the collision by dividing the desired velocity change by the velocity change per unit impulse. This gave me the value of the impulse int the direction of the contact normal. For each particle, I then applied a linear impulse (impulse*inverse mass) and a angular impulse (impulsive torque calculated by getting cross product of impulse and relative position, which is then multiplied by the inverse moment of inertia to get the change in angular velocity). This adjusted both velocities for each particle.

1.4.3 Moments of Inertia

For applying impulsive torque to polygons, I had to assign them a moment of inertia. The moment of inertia of an object defines how easy it is to start / stop spinning the object and is analogous to how mass affects acceleration or decelerating a object linearly. For rectangles I used the proper moment of inertia $\frac{1}{12}(w^2 + h^2)$ and for all other shapes I approximated them to a circle spinning around its centre which has the moment of inertia $\frac{1}{2}mr^2$.

1.4.4 Taking Shots

For taking a shot in my game, the player draws an arrow that represents a force on the polygon. This force has an origin, direction and magnitude and for the linear component I simply add the direction of the force multiplied by the magnitude and the inverse mass of the player to the linear acceleration. For the angular component I take the cross product of the relative position of the origin and the force and then multiply it by the inverse moment of inertia. Both these values have a max force that can be applied in order to stop the player applying velocities that break the collision detection.

1.4.5 Problems and Potential Areas for Improvement

While playing my game, you can see that every once in a while a shot will appear to bounce back much faster than it hit the wall. I think this is at least

partly due to not having an algorithm to pre-process contacts in order to remove multiple contacts between the same objects or at least resolve them in order of interpenetration. If a shape hits a obstacle flat on such that two points of the shape are penetrating the obstacle in the same frame, my contact resolver would resolve both contacts, generating angular changes in velocity which counteract each other and then generating a linear velocity change that is double, causing the particle to fly back from the wall quickly. I also could have improved my interpenetration resolution as discussed above and I would ideally have liked to implement friction into my game.

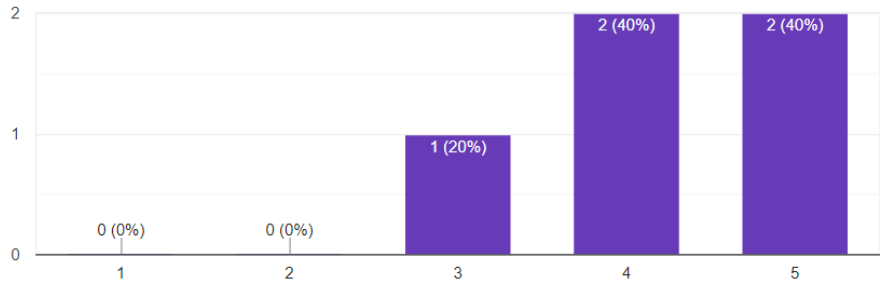
2 Context

My inspiration for this game has come from both playing actual golf and many golf video games. Playing the games Super Stickman Golf and Flappy Golf on mobile pushed me towards a game with simple controls and art style. The original twist to my game compared to other golf games is the addition of a randomly shaped polygon in place of the ball.

3 Evaluation

To test and evaluate my game I performed user testing. To do this I got other students to test my game and fill in a Google Form to evaluate their opinions on the game. Here are the four questions I asked an an evaluation of their answers.

3.1 Was the gameplay easy to understand?

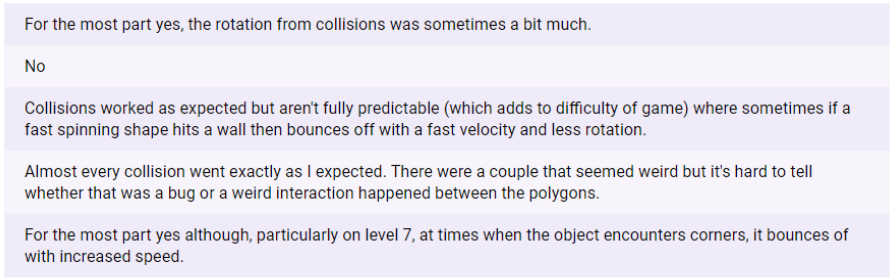


(1:Hard - 5:Easy)

I asked this question to find if my game needed more in game cues such as when you could take shots or if the minimalist design worked well . Based on the answers I decided to keep my design as the only user who scored a 3 was testing before the player guide was made and suggested that the controls were

fine once they had had a few shots, implying that they would be happy with them if they had had prior instruction on how to play.

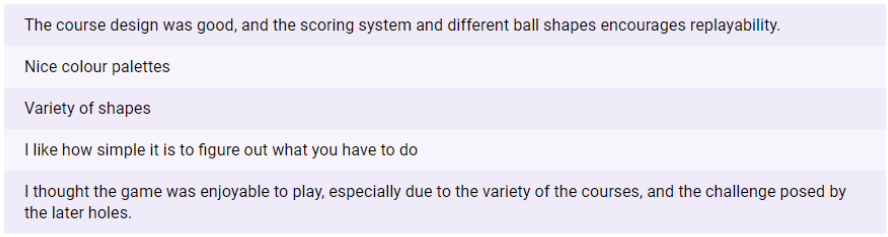
3.2 Did collisions behave as you expected?



Since I was using a rigid body collision system, I knew there were going to be problems with collisions and how they were resolved. I also knew from my game that sometimes the collisions could be frustrating and this is why I asked this question.

The answers were mostly positive with 4 out of 5 saying mostly yes but with everyone having some issues with collisions. I have discussed above some of the issues with my collision system and some of the collisions aren't behaving as they should. However I added a limit on the amount of angular force you can apply to a polygon when taking a hot as otherwise you could hit the polygon softly but with a massive amount of rotation and when it collided with the wall, this could be converted into linear velocity, causing frustration as your softly hit shot flew away. I think that upon playing the game more, you would learn to mostly try and hit the ball through the centre of mass in order to avoid excess spin.

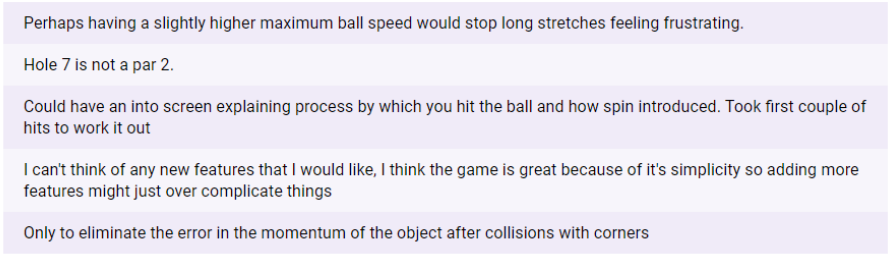
3.3 Was there anything in particular you liked about the game?



I asked this question to find out what the best parts of my game were to help me build up a picture of what parts of the game worked well. The users appeared to enjoy the hole designs which was good to hear and if I was making more holes, I would ensure to keep them in a similar style. One user said they

liked the variety of shapes and another said they enjoyed the colour palettes which is what prompted me to add the third, "sand", environment.

3.4 Were there any improvements you thought could be made to the game?



I asked this question to discover what the biggest flaws were in my game and to see if there were tweaks I could make to improve the game. The answers provided a few simple changes I was able to make to improve the gameplay. Firstly I increased my cap on the linear velocity due to taking a shot to improve taking longer shots as suggested. I had missed that the par for hole 7 was wrong and so after seeing the response about it I was able to increase it to 3. The question about the lack of instructions has been addressed by the creation of a player guide to teach the user how to play.

3.5 Other testing

Apart from user testing, while developing my collision code throughout the creation of my game, I had a single level that gave the polygon a constant velocity towards obstacles to help me test how the collisions behaved.