Anthony Rota - Logbook

Within the logbook:

- A list of work completed is to be identified such as:
    - *Completed the main menu function*
        - *Includes a login input and checks against a list of users*
    - *Added in some colours to the main menu function*
        - *Imported colorama library*

- Challenges or achievements:
    - ***Achievement:*** *Issues with user input have been solved by checking if it is alphabetical.*
    - ***Challenge:*** *Using the colorama library of code and attaching it to all documents to be submitted to work properly*

- References and links that are vital to the development of the project
    - *Geeks For Geeks - Dictionaries - https://www.geeksforgeeks.org/python-dictionary/*
    - *Images can be included to demonstrate the work completed or highlight issues*

Term/Week	**Work Completed**	**References / Links**
Yr11 Hol	Commit 1: init Starting writing code for the game. I decided to use pyglet instead of pygame because its api seems simpler, and its OpenGL support is better. I am implementing each screen with a class. An instance of a screen class can be created and assigned to the main Game instance, which will then update and render the screen each tick. I have a separate update loop inside the render method so that every second the active screen's update method is called a constant number of times (right now 120 times/second). (Ex: if there's a 1 second lag between renders, the update method will be called 120 times compared to the render method only being called once). This has the advantage of ensuring the physics updates are consistent, but the disadvantage that if the time between renders is a long time (eg. switching screens away from the game, then coming back after a few minutes), the update method will be called (120*number of seconds) times. I can fix this by detecting "equilibrium" states in the physics updates (eg. when an update does nothing because the golf ball	Pyglet documentation: https://pyglet.readthedocs.io/en/latest/

	is still), and returning a flag from the “render” method to communicate such an”equilibrium” state to the game instance. The only screen that renders anything right now is the menu screen, which has placeholder text to test that my screen logic works.	
T1 W1	Commit 1: feat: menu screen Added the Roboto font. Created a basic menu screen with buttons to play in levels mode, play in infinite mode, go to the options screen, and to quit the game. To do this I am using the glooey library which is a gui library for pyglet. Commit 2: feat: basic proc cave gen Implemented a basic cave generation algorithm. The algorithm has two stages: 1. In the first stage, I generate a two dimensional grid randomly filled with 0s and 1s assigned to each cell, with 0 representing an open space and 1 representing a wall. Then, I run the cellular automata procedure outlined in the roguebasin website a few times on the grid, which makes it look more like a cave. After this, I run a flood fill algorithm over the grid to detect the open spaces, and I extract the largest open space into its own grid. If this largest open space is big enough, then I return the grid as the generated cellular cave. Otherwise, I scrap the grid and keep trying to generate new ones until I find an open space big enough. 2. In the second stage, after the cellular cave is generated, I run the marching squares algorithm over the grid to generate the contours of the cave walls. Although the procedural generation is purely random right now, I plan to make it seeded. Made Gantt Chart	Glooey documentation: https://glooey.readthedocs.io/en/latest/ Cellular automata: http://roguebasin.com/?title=Cellular_Automata_Method_for_Generating_Random_Cave-Like_Levels#Rule_Tweaking Marching squares: https://en.wikipedia.org/wiki/Marching_squares
T1 W2	Commit 1: feat: render cave Implemented basic terrain rendering. First, I generate new contours using the shapely library (GEOS wrapper for python) by buffering the contours generated by the procedural cave generation algorithm. These buffered contours are essentially a scaled version of each contour, outwards for the outer walls and inwards for the inner walls. Then, I run these contours through the GLU tessellation algorithm to triangulate them, and I store the vertices and indices generated in buffers to later render. I also create a pseudo 3d effect by looping through each edge of each cave contour generated by the procedural cave generation algorithm, and detecting if the edge is on the upwards face of the contour, meaning that it is a “ground” edge. For each “ground” edge, I extend it upwards to create a rectangle that looks like it goes into the “z”-axis, creating a pseudo 3d effect.	Shapely documentation: https://shapely.readthedocs.io/en/stable/manual.html GLU tessellation guide: https://www.glprogramming.com/red/chapter11.html

T1 W3	Commit 1: feat: camera Implement a camera class which has a (x, y) position and a width (describing the view frame), which along with the aspect ratio of the window determines what on the screen is in view. Now, when the camera's view frame is larger than the bounding box of the cave, I subtract the cave's bounding box from the camera's view frame to create a list of rectangles (where before nothing was rendered and therefore here was black) to render the cave's background. Fixed render loop such that it is called "fps" times per second, whereas before it was only called during interactions with the window. Commit 2: refactor: split geometry from screen class Separated the rendering of the golf levels from the screen class. To do this I implemented a LevelGeometry class which takes in the contours and configuration for the level, generates all of the geometry needed to render the level, and has a render method that renders the level. Then, in the screen class, I create an instance of the LevelGeometry (eg. in PlayInfiniteScreen, I pass in the randomly generated cave contours as an argument), then I just need to call geometry.render(camera). I also split many of the utility classes into their own files. Commit 3: feat: start and end places Randomly generate the start place where the golf ball starts, and the end place where the flag is placed. First, I make a list of all the flat places in the contours. Then, I find the "best" pair of flat places by a heuristic which sums the length of the potential place where the flag could be placed with the length of the minimum path from the potential start place to the potential end place. Thus, the returned pair of (start_place, end_place) is far apart, and the flag is placed in a flat area that is wide.	Pyglet camera example: https://github.com/pyglet/pyglet/blob/7a0be60f1dc5d605689cec7d46169e72c6762dba/examples/window/camera.py
T1 W4	Commit 1: fix: improve start and end placement Removed the requirement for the flat places where the start and end positions are chosen to be of a minimum width, as the heuristic already accounts for the width of the place where the flag is placed, and the width of the start position isn't of importance. Also fixed the A* algorithm where the (x, y) indices were switched around. Commit 2: feat: shaders and stripes Updated rendering so that it uses shaders instead of some obsolete OpenGL methods. I ran into a lot of segfaults that took a while to debug (one was that I was letting OpenGL calculate the screen space positions of the vertices in the vertex shader during testing for me, and then later I calculated	Physics numerical integration methods: https://research.ncl.ac.uk/game/mastersdegree/gametechnologies/previousinformation/physics2numericalintegrationmethods/2017%20Tutorial%202%20-%20Numerical%20Integration%20Methods.pdf

	them manually with the passed in view matrix, but I didn't remove the former code). Another one I allocated double as the indices buffer type rather than uint. I also implement a stripe shader to render the flat terrain around the flag with stripes. To do this I calculate the distance from the vertices to some slanted line and then depending on this distance I render the background colour or the stripe colour. I need to fix the stripe shader though as if the striped terrain passes through the slanted line, then the stripe will be twice as large (as the distance I calculate to the line is not signed). Commit 3: fix: stripe shader near line Fixed aforementioned wrong behaviour of stripe shader. Commit 4: fix: don't round contours as it sometimes breaks To round the contours, I buffer them to make them bigger than buffer them to make them smaller which rounds the vertices, but this creates issues when the buffered contours are close together as they probably intersect which breaks the geometry. Now I don't round the contours. Commit 5: feat: start implementing physics Created Physics class that updates the ball's position. It doesn't handle collisions yet. Commit 6: refactor: LevelGeometry.py -> Geometry.py Renamed file. Commit 7: fix: make ball sprite use float coordinates The position of the ball on the screen was janky because it's position was rounded off to the nearest pixel (the janky effect exemplified by the fact the image is scaled up) so to fix this I enable the pyglet sprite to use subpixel coordinates. Commit 8: refactor: asset loading Now asset loading is encapsulated in a singleton class. Commit 9: fix: group shader render together Fix render code so that all the objects for a shader are rendered together meaning the shaders don't have to be unnecessarily bound and unbound.	Pymunk documentation: http://www.pymunk.org/en/latest/

	Commit 10: feat: basic physics Use the pymunk physics library to handle collisions between the ball and the terrain. Commit 11: feat: drag to fire Implement the drag to fire functionality. When the user drags back their mouse and releases, the ball’s velocity is pointed in the direction of the vector from the mouse’s end position to the mouse’s start position. However this is buggy right now as sometimes pyglet doesn’t recognise the mouse release event (I think this is a pyglet bug? See: https://github.com/pyglet/pyglet/issues/171). I might have to have a separate button to release the ball if I can’t fix this buggy behaviour. I also generate a preview of the shot by simulating the path of the ball if it were to be released, which is rendered as a line but I will make it dotted. UPDATE: the buggy behaviour only occurs when using the macbook trackpad, I think it’s an issue out of my control? Using a mouse, it works fine. Also, there's a pyglet bug where every so often when opening the game, the window doesn’t receive any events, and the only way to receive events is to switch the window out of focus and back in focus. See: https://github.com/pyglet/pyglet/issues/225. Commit 12: feat: make shot preview dotted Implement a dotted line shader which works by discarding the pixels which are not part of the dotted line. I also refactored the buffer handling into its own separate class so I could use this to make the buffers for the dotted line. Commit 13: feat: make shot handling better I made the shot preview line fade out near the end so it doesn’t suddenly stop. I now track the state of the ball: i.e. whether it is in motion or stationary, and count shots taken. I also detect when the ball has collided with the flag through a collision handler between the ball and the flag objects. However right now nothing happens when the ball gets to the flag. I also lock the shots so that a shot can only be taken when the ball is stationary.	
T1 W6	Made initial data flow diagrams - will update continually when I add more stuff.	
T1 W11	Commit 1: feat: polygon shot preview The shot preview now previews a “range” that the ball can land in by rendering a polygon outlined by two dotted lines, instead of just a single dotted line representing the exact path the ball will take. To do this, I simulate two paths of the ball, one with a slightly increased vertical velocity and one with a slightly decreased vertical velocity, then generate triangle strips using the two paths to	

	generate a polygon that is rendered transparently. Commit 2: fix: tweak path preview Made one of the simulated paths of the ball start at the top of the ball, and the other at the bottom of the ball, instead of both starting at the middle of the ball, because I think it looks better.	
Hol W1	Commit 1: feat: make infinite mode infinite When the ball collides with the flag I call the level complete callback passed to the Physics class, wherein the active screen is set to the next level. The next level is pre-loaded in a separate thread, but there is slight lag as the geometry has to be generated for the next level. Commit 2: feat: ball trail Implemented a ball trail. To do this I store the last X positions of the ball, then render a transparent polygon through these points which fades out near the tail. Commit 3: feat: sand pits Implemented randomly generated sand pits. To generate the sand pits, I first locate all the potential positions of the sand pits. To do this, I walk around the outer cave wall, and detect when the wall dips down then dips back up, and store these as potential places for the sand pits. I also filter these locations such that there is no terrain inside them (i.e. the potential place for the sand pit does not intersect with any terrain), and they are not located on the start or flag green. I also filter them such that their areas are within a certain range so they are not too small or too big. Then, I randomly choose up to N random sand pits from these potential places for the sand pits. Finally, for aesthetics I add a sine wave on the top of the sand pit to make it look wavy. Commit 4: feat: sticky walls Implemented sticky walls, which the player can place wherever they want. Currently, pressing the “G” key toggles from “sticky” mode to “normal” mode (“G” for Goo?). In “normal” mode, the game is as normal and the player can make shots. In “sticky” mode, the player can’t make shots but instead can make the walls sticky. When a wall is sticky and a ball hits it, the ball gets stuck to it. In this way the ball can become stuck on the walls and can reach places it couldn’t previously. When the player is in “sticky” mode and hovers over the edge of terrain, it previews which walls will become sticky by rendering them as sticky but slightly lighter. When the player left clicks, these walls turn sticky. To figure out which walls will turn sticky, I find which parts of the contour are within a certain radius of the mouse, then get the closest continuous strip of such parts of the contour,	Pyglet event docs: https://pyglet.readthedocs.io/en/latest/programming_guide/mouse.html Python threading help: https://stackoverflow.com/questions/6893968/how-to-get-the-return-value-from-a-thread-in-python Pixel art in photoshop tutorial: https://www.youtube.com/watch?v=rLdA4Amea7Y Easing functions: https://easings.net/

which is what will be turned sticky. When the player left clicks, these sticky walls are stored and geometry is generated for them. A collision handler is added to the corresponding walls in the Physics class, which makes the ball stop when it collides with them. For optimisation and also to join segments of sticky walls together that are placed next to each other, separately, when a sticky segment of wall is added it checks if it is next to any other sticky segments of wall, and if so joins them. Note that walls which are next to sand pits and also parts of the contour that are horizontal ground cannot be turned sticky. I also have to add that my circle-line segment collision detection function worked on the first try!

Commit 5: feat: tweak sticky walls
Added stripes to the sticky walls, and also made it so they protrude slightly from the walls.

Commit 6: fix: group sand pit geometry generation
Grouped together the code that generates the OpenGL geometry for the sand pits.

Commit 7: fix: check left click for placing sticky walls
Fixed a bug where mouse clicks that were not left clicks also placed sticky walls. Now only left clicks place sticky walls.

Commit 8: feat: cancel shot
Allow the player to cancel their shot by clicking either escape, “c”, backspace or delete.

Commit 9: feat: limit max power
Limited the maximum velocity of the ball as it is shot and also adjusted the sensitivity of shooting.

Commit 10: fix: allow shot when start dragging before able to shoot
When the ball has been shot and is still moving, the player cannot take another shot until the ball is stationary. This commit allows the player to hold left click before the ball becomes stationary, then when the ball is stationary if the user moves their mouse (while still holding left click), the shot preview will be rendered and the shot can be taken.

Commit 11: feat: drawback circle
Render a circle to show the user where they have drawn back from and how far they have drawn back while they are aiming to take a shot. The circle’s centre is where the user started dragging

back from, and it goes through the mouse (unless the power of the shot is limited then the radius is restricted). In this way it is easier for the player to navigate their mouse to aim their shot.

Commit 12: fix: improve mouse handling and new level lag
Make mouse handling more intuitive and make the cancel keys also cancel placing a sticky wall. Also, now the current frame is rendered before the new level is loaded (which takes a bit), instead of after, so there is not a flicker right before switching to the new level.

Commit 13: feat: make ball a solid circle with outline
The ball is now a generated solid circle with a thin outline, instead of an image.

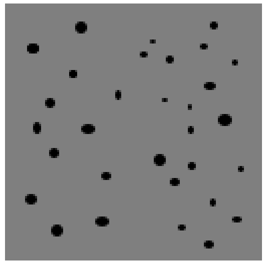
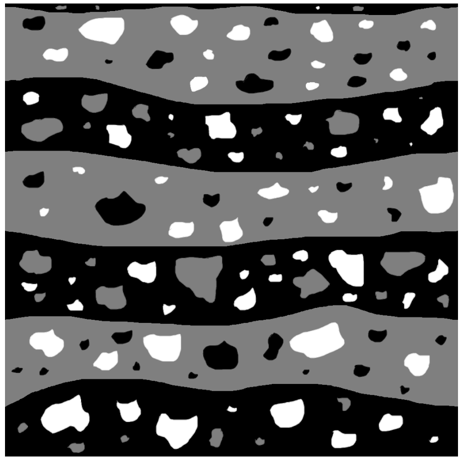
Commit 14: fix: canceling sticky mode
Make the cancel keys switch back to make shot mode from place sticky wall mode.

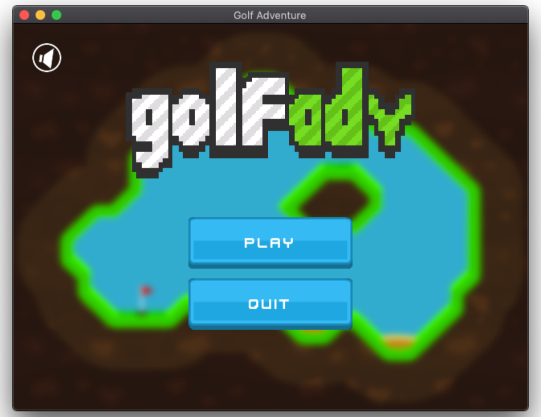
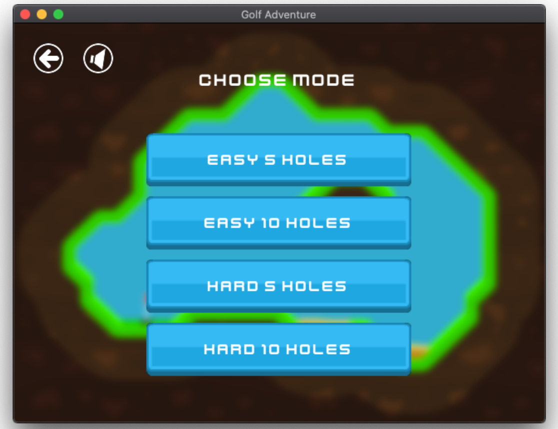
Commit 15: fix: make sure ball can squeeze through sand pit/contour gap
When playtesting I came upon a cave where the generated sand pit was too close to one of the floating wall things, and the ball couldn't squeeze through the gap. Now in the collision detection to detect whether a candidate sand pit is "good", I inflate its collision polygon by the radius of the ball to ensure that the ball can fit through such gaps.

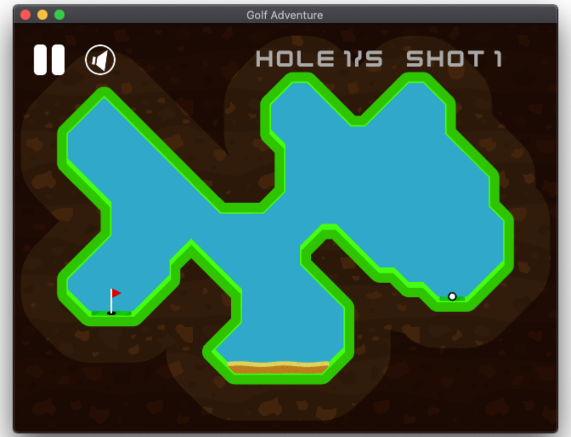
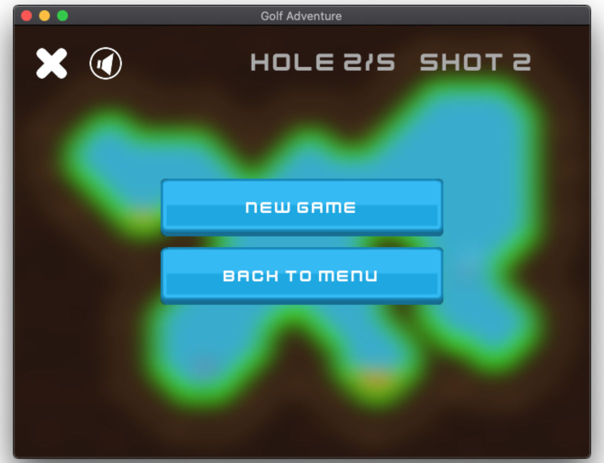
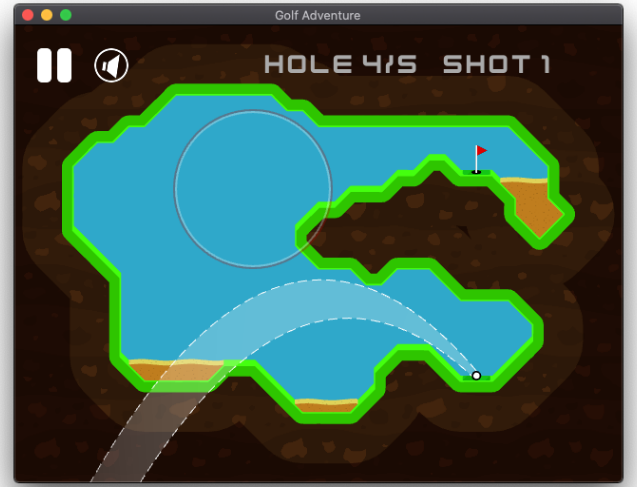
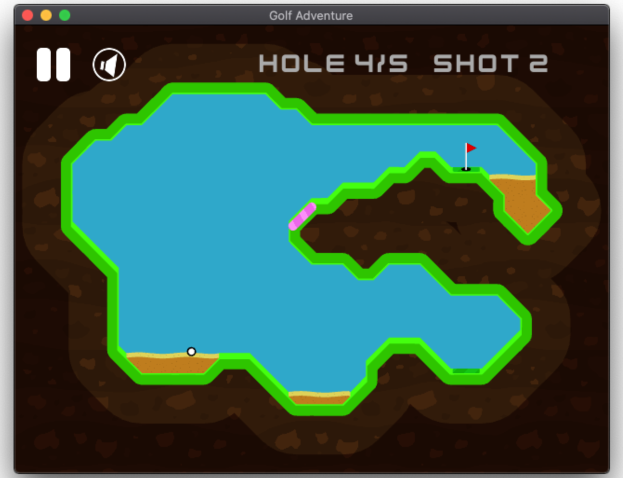
Commit 16: fix: mouse handling when switching modes
If the player switches to shoot mode while left click is already held down, then now they have to release left click and press left click again in order to attempt a shot (instead of it immediately going into a shot attempt). Same with switching to place sticky mode while left click is already held down - the player now has to release and press left click in order to place a sticky (whereas before if they released left click when it was held down before switching, it would place a sticky)

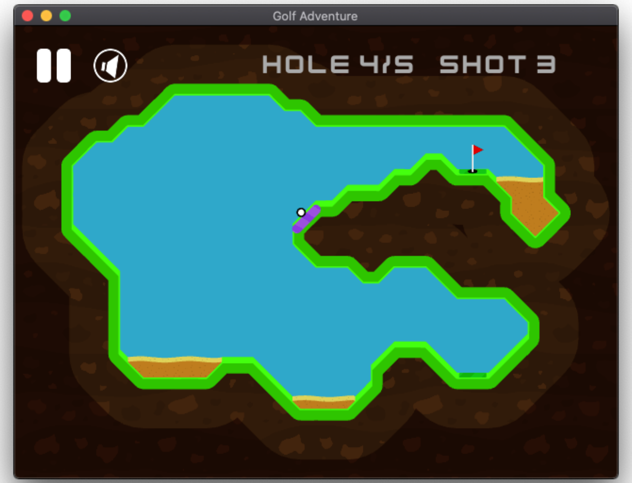
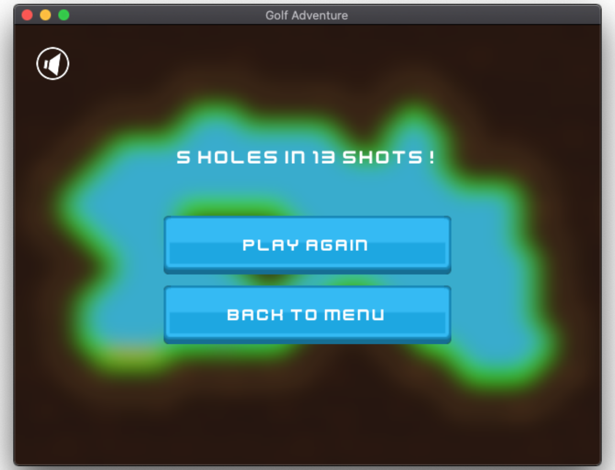
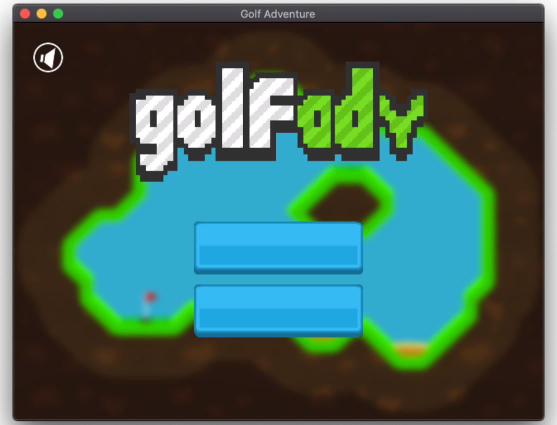
Commit 17: feat: flag
Implemented a flag. The flag is loaded as an image and rendered as a sprite. A challenge was that the hole in the flag image is not at the centre of the image, so I had to be careful when positioning the flag sprite to make sure the centre of the hole is on top of where the game positions the hole.

Commit 18: feat: ball into hole animation
Fixed the flag physics so that the collision between ball and the flag is only detected when the ball

	touches the hole, whereas before it was a much larger radius of collision. Now, when the ball touches the hole, it is quickly animated into the hole. FIrst, it is animated directly on top of the hole, then it is animated to fall down the hole.	
Hol W2	Commit 1: feat: textured sand Added sand texture.  The texture is read in the sand shader and the black parts darken the sand. Commit 2: feat: texture dirt Added dirt texture.  Uses the same shader as the sand but the white parts lighten the dirt. Commit 3: fix: flickering when resizing window Now when the window is resized, the game is immediately re-rendered. Before the screen would flicker black when resizing as it would be temporarily cleared.	OpenGL texture tutorial: https://learnopengl.com/Getting-started/Textures

	Commit 4: fix: render correct number of wall rectangles Forgot to add in the number of triangles to render when refactoring the code. Added it back.	
T2 W1	Commit 1: feat: gui and stuff Made a bunch of assets. Added a proper GUI. The menu screen has buttons to play the game, and to quit the game. There is a sound toggle at the top left of every screen. Clicking to play the game takes to the game mode screen where there are buttons for the four game modes (and a back button) - easy (5 holes), easy (10 holes), hard (5 holes), hard (10 holes). While playing the game there is a pause button which pauses the game, and overlays buttons to start a new game or go back to the menu screen. At the top right the hole number and shot number is displayed. I made a shader that blurs the game in the background when the game is paused (and animates this blur in and out). When the game is finished, a message is displayed with the score (number of shots) and buttons to start a new game or go back to the menu.   Golf Adventure CHOOSE MODE EASY 5 HOLES EASY 10 HOLES HARD 5 HOLES HARD 10 HOLES	Glooey library: https://github.com/kxgames/glooey OpenGL Blur: https://github.com/Jam3/glsl-fast-gaussian-blur Pyglet sound documentation: https://pyglet.readthedocs.io/en/latest/programming_guide/media.html

	 Golf Adventure HOLE 1/5 SHOT 1  Golf Adventure HOLE 2/5 SHOT 2 NEW GAME BACK TO MENU  Golf Adventure HOLE 4/5 SHOT 1  Golf Adventure HOLE 4/5 SHOT 2	

	  Golf Adventure 5 HOLES IN 13 SHOTS! PLAY AGAIN BACK TO MENU Challenge: sometimes the text for all the buttons in a screen are not rendered. I have no idea why. If I then switch screens and go back the text appears as it should.  Challenge: Very occasionally clicking to start a game causes a crash because of a segfault. I don't know what is causing this. Commit 2: [feat: sound] Added sound. I made the background music and some sound effects. The rest of the sound effects	

	I took from a youtube channel/website that says they are copyright free and don’t need attribution!!!!!!! Sound effects for: clicking button/icon, making shot, ball hitting sand, ball hitting goo, and the ball going in the hole. Updated dataflow diagrams.	