**Bridgewater State University**

[COMP 510-001 Topics: Rust vs Go](https://bridgew.blackboard.com/webapps/blackboard/execute/launcher?type=Course&id=_52305_1&url=)

**Submitted by:**

Saumya Bhatnagar

**Banner ID:**

00370168

**Submission date:**

19th December 2018

Rust vs Go: The final

Both Rust and Go are powerful modern-day languages that are growing fast. After spending a semester learning both these languages, I have found Go to be my language of preference. Go is more intuitive and easily understandable as compared to Rust, which is why I am using this language for my final project.

For the final project of the semester, in the spirit of Old School Renaissance movement, I have decided to work on the Atari game: Berzerk.

A small description of my project:

* The project has four levels.
* The project has a player that is controlled by the user. The player can move in all directions using the arrow keys, and shoot bullets using the space bar.
* At the start of the project, the player has three lives, and zero score.
* The player can shoot in four directions and bad guys can shoot in eight directions.
* There are bad guys in each level that shoot randomly at the player.
* The player loses a life if any of the below event occurs:
  + Player is hit by a bad guy’s bullet
  + Or player touches a wall
  + Or player touches a bad guy.
* The player’s score increases by ten points each time a bad guy dies. The bad guy dies either by getting hit by player’s bullet or when bad guy touches a wall.
* Once all the bad guys in the current level are dead, the player can proceed to next level through one of the openings.
* If the player reaches the fourth level and is able to finish all the bad guys either by shooting at them or by bad guys hitting a wall, the player wins and the game is over.
* If the player loses all 3 lives, it is ‘game-over’ for the player.

To run this game, below libraries have to be imported:

|  |
| --- |
| pixel: This library is used to import the settings for the GUI window used by the game |
| beep: This library is used to import the sound functionality in the game |
|  |

Use the go get command to import these libraries:

|  |
| --- |
| go get github.com/faiface/pixel/… |
| go get github.com/faiface/beep/… |
|  |

The project can be run using any IDE. I have used Goland by Jetbrains. It can also be run by the command: go run main.go from the command line.

The first step to begin the implementation of the project was to search for a gaming library in go. The github page[[1]](#footnote-1) lists the game development libraries available with Go.

In my quest to find a library that is easy to work with, I considered Ebiten, go-sdl2 and Pixel. From these options, I selected the Pixel library because of its documentation and available examples. Next, I followed the instructions in the tutorial to understand how the pixel library works by creating a *hello-world* equivalent, which is a blank GUI window. Once this step is done, I tried creating a few sprites and images on the screen.

Pixel uses a run function where all the functionality is performed. This run function is then called by main. For the first level, I created the walls using IMDraw. The walls are essentially lines which are created by giving a start point(x,y) and an end point(x,y). The decision to use lines as walls for my project turned out be one of the pain points during the project implementation. The sprites for player and bad guys are randomly placed on the screen at legal positions defined by an x and y coordinates. This (x,y) position is the center of the sprite. To check if the sprite touches a wall, and the wall being a series of x and y points, almost half sprite has to overlap with the wall for their x,y position to match. As a result, the effect of bad guy dying upon touching the wall is diminished. To assuage this discrepancy, I have added and subtracted half the length and width of my sprites in the function before the comparison. Below code snippet best explains the idea:



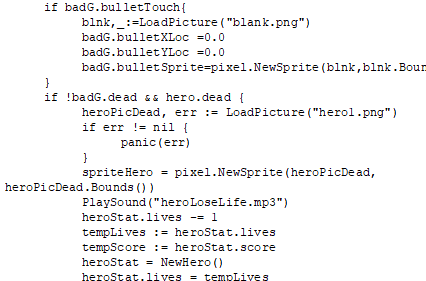
The game has 4 levels. Forgetting all the software engineering principles, I coded majority of the functionality in the run function. This proved to be a mistake as now, I could not make the levels change from run function. Essentially, my program was a giant run function that has a series of commands running sequentially. After finishing level, I realized I would have to break the programs into separate functions to be able to call them in run for different levels. At this point it was easier to write the entire program from scratch than to refactor the existing.

As specified in the instructions, if the player manages to kill all the bad guys, the player has to leave the map through one of the openings to move on the next level and begin the next level at that point. However, the design of the walls of my levels did not support this functionality. Every time the player tries to go to next level from the x,y positions of the current level that are part of the wall in next level, the player loses a life. To avoid losing a life here, I have made the player spawn at a random legal position when the level changes.

Another functionality directs, that when the player loses a life by getting hit a bullet from bad guy, the player should lose a life and bullet should be removed from the screen. To achieve this, at first I was making the bullet “disappear” by changing the bullet sprite with a blank image. But I realized soon that this is not effective. If the player re-spawns at a legal position in the path of the bullet, then the player gets killed again frim the invisible bullet. To remedy this, instead of changing the bullet sprite, I changed the bullet x and y position to (-500,-500). However, this means if the player stays at the level long enough even after the bad guys are dead, the bullet will reappear on the window, and if it hits the player, the player loses a life again. At the end, I made the below changes to get the desired effect:

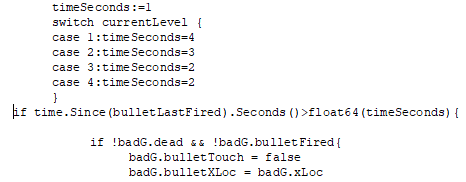
* + Change the bullet sprite to blank once the bullet hits the player
  + Change the position of the bullet to make it off the screen
  + Put a check on the bad guy that shot the bullet. If the bad guy that shot the bullet is dead, his bullets cannot kill the player.

Below is a code snippet for the same:



There are unique sounds for different events like player proceeding to next level, player shooting, bad guy shooting, player losing a life, bad guy losing a life, game over, and player win. These sounds are played by calling a function PlaySound from a third party library github.com/faiface/beep. When the function is called, it plays the sound clip from the file specified in the path. However, this causes the game to pause for the duration of the sound clip. To minimize this effect, I have used small sound clips, but the lag becomes noticeable at higher levels where the bad guys shoot more frequently at the player. Although I was not able to resolve this issue before the project submission, one way to resolve this issue is to make use of go-routines and channels for playing the sound.

As per the instructions, the bad guys should have a random chance for one bad guy to shoot at the play each second. For the project, I have made the frequency of bad guys shooting at the player a function of levels. As the levels increase, the time elapsed between bad guys shooting decreases. Below snippet explains the idea.



A functionality I would like to add in the game is, as the level increases, the number of bad guys shooting at the player should also increase.

Apart from the above mentioned points, the remaining functionality was achieved majorly by the use of if-else conditions and switch statements.

1. https://github.com/avelino/awesome-go#game-development [↑](#footnote-ref-1)