

Team 35 - Terminal Invaders

Jasper Calder, Ryan Chang, Jason Zhang

Team Objectives

Project:

- Meet the project requirements and criteria (as specified in the project guidelines)
 - Using modern c++, demonstrating computation thinking, using collections, uniform initialization, implementing exception handling etc.
- Code is modular and can be easily modified or expanded upon
- (developing code incrementally, allowing the ease to test and debug each new feature)
- Attempt to have equitable contributions from team members
- Incorporate comprehensive documentation with comments to clearly communicate to the viewer justification for how we chose to program our game components
- Utilize or be inspired by and build on top of the sample code provided initially

Interactive Experience:

- Design an interactive and bug-free experience for the player
- Intuitive controls that users can easily understand and use to play the game
- Add specific features such as score and restart functionality in order to make the game more compelling for users to play

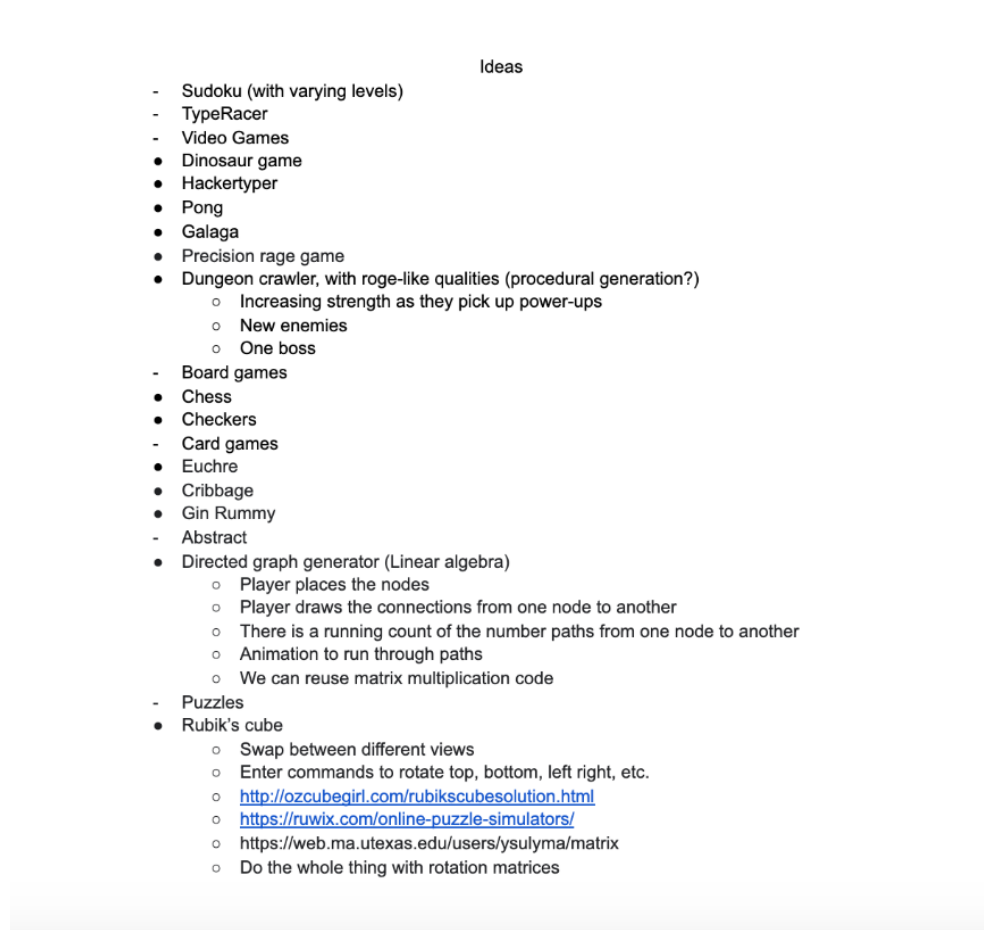
Design Process

Our team first went through Professor Foster's sample programs and played around with it to gain a better understanding of the different functions such as displaying characters, reading input, and setting a screen on the terminal.

After becoming more familiar with the base code, we started to brainstorm the types of interactive experiences we could create. Our team unanimously voted to go with a game project as they are engaging for users and is something that we would enjoy building as well.

We wanted to come up with a game concept that would be challenging, yet feasible in the given time constraints. With this in mind, our team came up with many different game concepts, based on online research, our own ideas, and experiences with games in the past.

We recorded some of the ideas in a document as shown below:



The team then discussed concerns with each idea and estimated the amount of work, as well as detailing, that would be required for the project.

Among our concepts were ideas inspired by retro video games that we have previously played, including Pong, Galaga, Space Invaders and Dungeon Crawler.

Ultimately, we decided on designing a Space Invader like game called "Terminal Invaders". We also prepared a backup plan of creating a version of Sudoku if we were not able to complete the original concept, as we felt that it would be easier to implement during time-constrained situations.

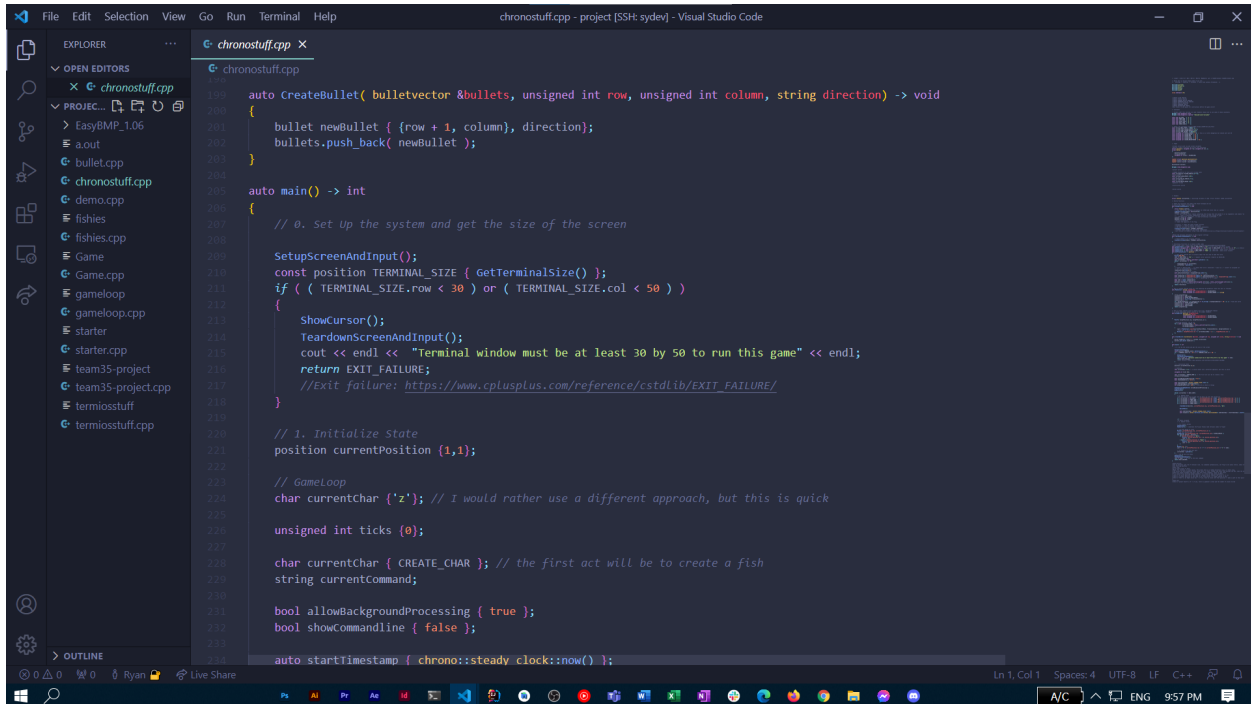
A large portion of our time was spent understanding and repurposing the fishies.cpp starter code. Since the architecture of fishies.cpp was not designed by members of the team, processing and understanding how each function connected with each other required a lot of trial and error in “playing” with different implementations. This resulted in many initial failed attempts in achieving basic actions namely, creating sprites, reading user input, as well as setting bounds for game objects. However, once we were familiar with the starter code, building additional features, such as game object structures, boundaries, and functions managing the state of the player, enemies, as well as lasers, became a simpler process.

A particular challenge we faced during the middle stages of design that we would like to highlight is enabling the enemies to move from one end of the game board to the other. The desired concept was that the enemies would initially move to the right until it hit the wall—the right-most column of the matrix representing the game board. Once the enemies arrived at the right-most column index, they would “bounce” off the wall and start moving to the left. Once they hit the left wall, they would respectively bounce off the wall and start moving to the right. Due to the fact that we had to iterate through the vectors one alien at a time, there was a challenge in getting a column of enemies to bounce off at the same time. This was initially approached by searching for the maximum and minimum column values and checking whether the maximum was equal to the right border size or the minimum was equal to the left border size. If either of these conditions were true, the enemies would change direction. However, since the enemies were already appended to the vector in increasing column order, the loop was not necessary. Instead of checking the maximum and minimum values, we could just check the beginning and end of the enemies vector.

After the main functions of the game were completed, the team decided to implement features such as a score counter and a restart option that would make the game even more enjoyable.

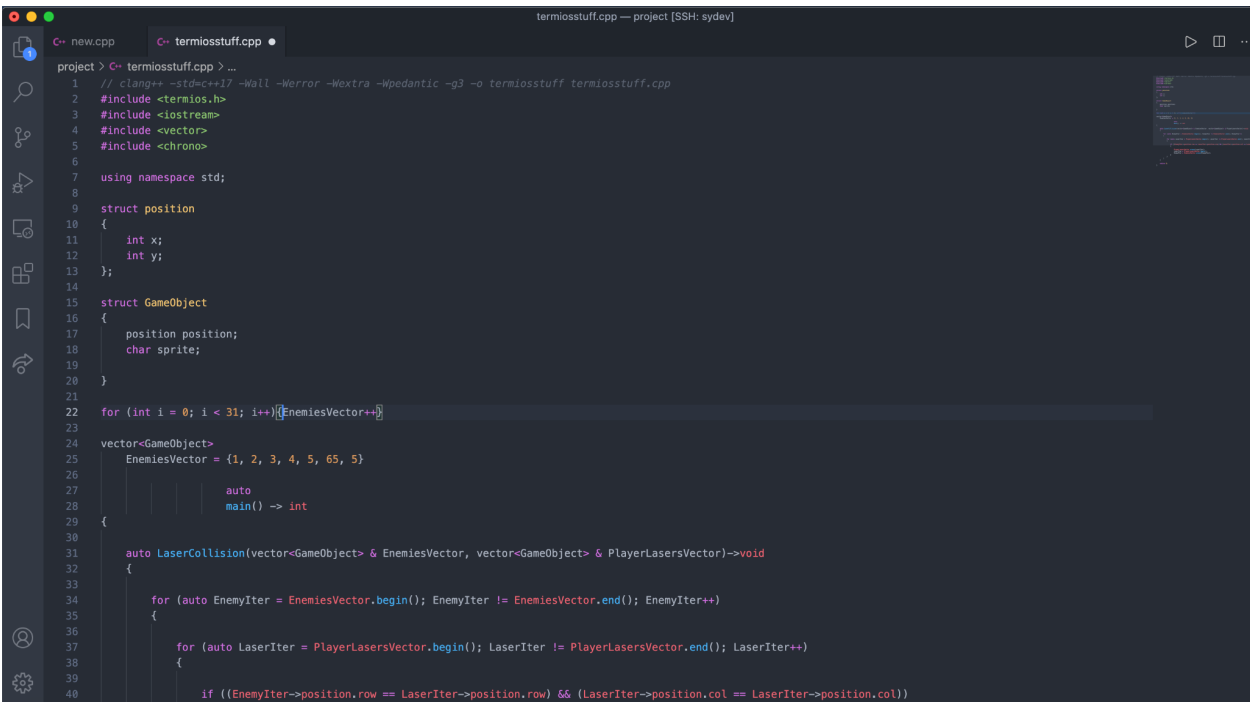
As a group, we are incredibly proud to not only fulfill the basic requirement of this project, but be able to create a robust product that we are proud of. Through teamwork, effective communication despite online constraints, and willingness to push beyond our comfort zones, Team 35 was able to make a great game that anyone can enjoy.

Iterations: The figures below illustrates a timeline of our design process



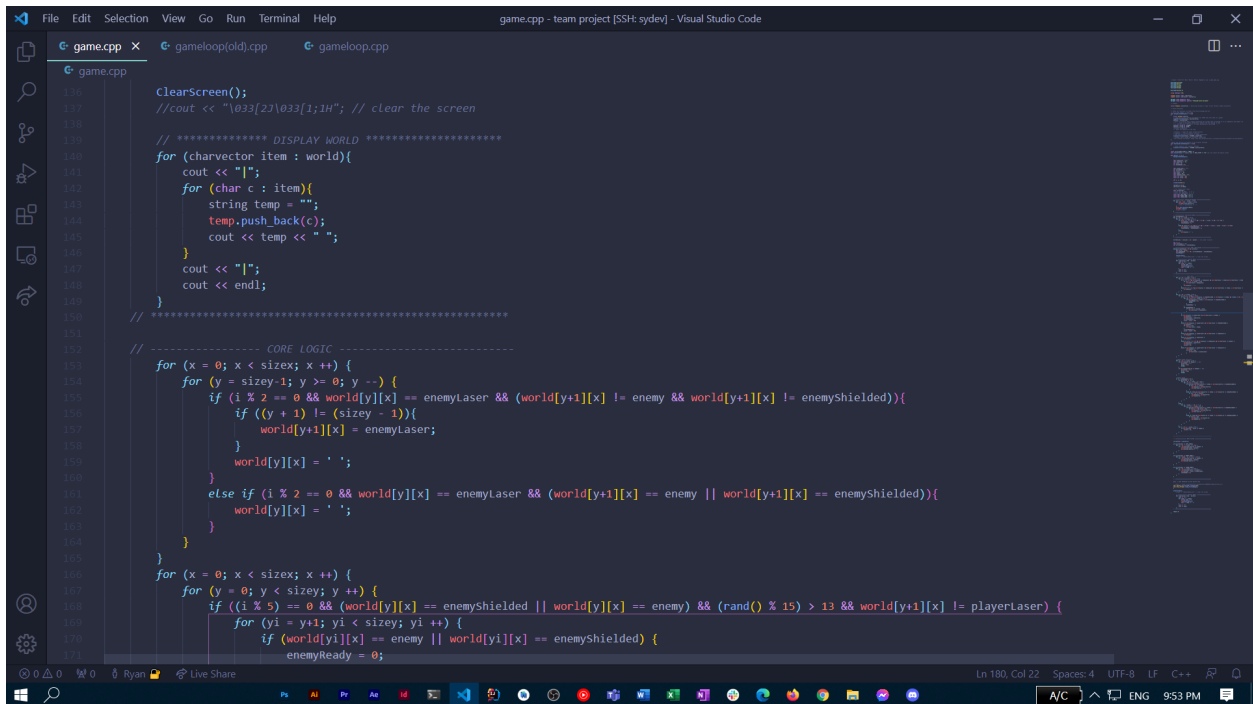
```
1 auto CreateBullet( bulletvector &bullets, unsigned int row, unsigned int column, string direction ) -> void
2 {
3     bullet newBullet { (row + 1, column), direction};
4     bullets.push_back( newBullet );
5 }
6
7 auto main() -> int
8 {
9     // 0. Set Up the system and get the size of the screen
10
11     SetupScreenAndInput();
12     const position TERMINAL_SIZE { GetTerminalSize() };
13     if ( ( TERMINAL_SIZE.row < 30 ) or ( TERMINAL_SIZE.col < 50 ) )
14     {
15         ShowCursor();
16         TeardownScreenAndInput();
17         cout << endl << "Terminal window must be at least 30 by 50 to run this game" << endl;
18         return EXIT_FAILURE;
19         //Exit failure: https://www.cplusplus.com/reference/cstdlib/EXIT\_FAILURE/
20     }
21
22     // 1. Initialize State
23     position currentPosition {1,1};
24
25     // GameLoop
26     char currentChar {'z'}; // I would rather use a different approach, but this is quick
27
28     unsigned int ticks {0};
29
30     char currentChar { CREATE_CHAR }; // the first act will be to create a fish
31     string currentCommand;
32
33     bool allowBackgroundProcessing { true };
34     bool showCommandline { false };
35
36     auto startTimestamp { chrono::steady clock::now() };
37 }
```

Fig 1. chronostuff.cpp. Initial program written to play around with the chrono features in fishies. This ultimately enabled the team to understand how to refresh the screen to update the game board in the final iteration.



```
1 // clang++ -std=c++17 -Wall -Werror -Wextra -Wpedantic -g3 -o termiosstuff termiosstuff.cpp
2 #include <termios.h>
3 #include <iostream>
4 #include <vector>
5 #include <chrono>
6
7 using namespace std;
8
9 struct position
10 {
11     int x;
12     int y;
13 };
14
15 struct GameObject
16 {
17     position position;
18     char sprite;
19 }
20
21
22 for (int i = 0; i < 31; i++){EnemiesVector++;}
23
24 vector<GameObject>
25     EnemiesVector = {1, 2, 3, 4, 5, 65, 5};
26
27     auto
28     main() -> int
29     {
30
31         auto LaserCollision(vector<GameObject> & EnemiesVector, vector<GameObject> & PlayerLasersVector)->void
32         {
33
34             for (auto EnemyIter = EnemiesVector.begin(); EnemyIter != EnemiesVector.end(); EnemyIter++)
35             {
36
37                 for (auto LaserIter = PlayerLasersVector.begin(); LaserIter != PlayerLasersVector.end(); LaserIter++)
38                 {
39
40                     if ((EnemyIter->position.row == LaserIter->position.row) && (LaserIter->position.col == LaserIter->position.col))
41                     {
42
43                     }
```

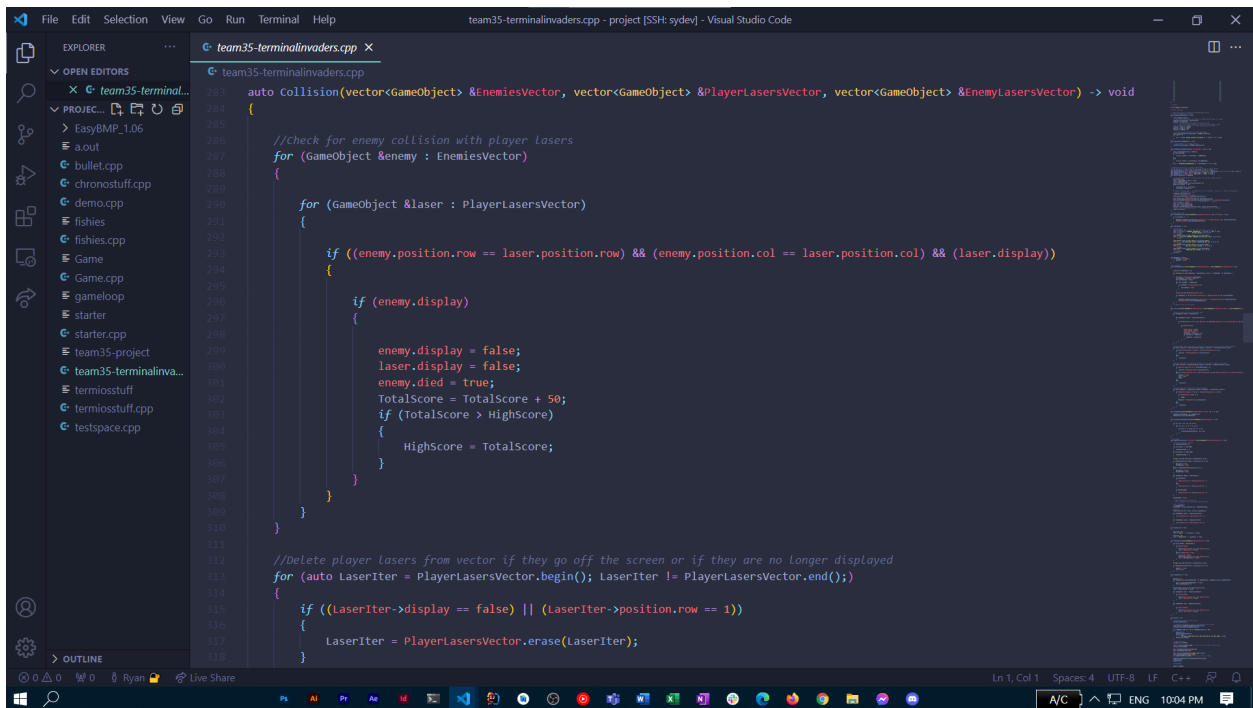
Fig 2. termiosstuff.cpp. Initial tweaking of fishies.cpp and adding a position + GameObject structure.



```
File Edit Selection View Go Run Terminal Help
game.cpp - team project [SSH: sydev] - Visual Studio Code

game.cpp
136 ClearScreen();
137 //cout << "\033[2J\033[1;1H"; // clear the screen
138
139 // ***** DISPLAY WORLD *****
140 for (charvector item : world){
141     cout << "|";
142     for (char c : item){
143         string temp = "";
144         temp.push_back(c);
145         cout << temp << " ";
146     }
147     cout << "|";
148     cout << endl;
149 }
150 // *****
151
152 //----- CORE LOGIC -----
153 for (x = 0; x < size; x++) {
154     for (y = size-1; y >= 0; y--) {
155         if ((i % 2 == 0 && world[y][x] == enemyLaser && (world[y+1][x] != enemy && world[y+1][x] != enemyShielded))) {
156             if ((y + 1) != (size - 1)) {
157                 world[y+1][x] = enemyLaser;
158             }
159             world[y][x] = ' ';
160         }
161         else if ((i % 2 == 0 && world[y][x] == enemyLaser && (world[y+1][x] == enemy || world[y+1][x] == enemyShielded))) {
162             world[y][x] = ' ';
163         }
164     }
165 }
166 for (x = 0; x < size; x++) {
167     for (y = 0; y < size; y++) {
168         if ((i % 5) == 0 && world[y][x] == enemyShielded || world[y][x] == enemy && (rand() % 15) > 13 && world[y+1][x] != playerLaser) {
169             for (yi = y+1; yi < size; yi++) {
170                 if (world[yi][x] == enemy || world[yi][x] == enemyShielded) {
171                     enemyReady = 0;
172                 }
173             }
174         }
175     }
176 }
```

Fig 3. game.cpp. Previous iteration of core game logic. Ultimately repurposed to add a greater level of modularity to the product.



```
File Edit Selection View Go Run Terminal Help
team35-terminalinvaders.cpp - project [SSH: sydev] - Visual Studio Code

team35-terminalinvaders.cpp
283 auto Collision(vector<GameObject> &EnemiesVector, vector<GameObject> &PlayerLasersVector, vector<GameObject> &EnemyLasersVector) -> void
284 {
285     //Check for enemy collision with player lasers
286     for (GameObject &enemy : EnemiesVector)
287     {
288         for (GameObject &laser : PlayerLasersVector)
289         {
290             if ((enemy.position.row == laser.position.row) && (enemy.position.col == laser.position.col) && (laser.display))
291             {
292                 if (enemy.display)
293                 {
294                     enemy.display = false;
295                     laser.display = false;
296                     enemy.died = true;
297                     TotalScore = TotalScore + 50;
298                     if (TotalScore > HighScore)
299                     {
300                         HighScore = TotalScore;
301                     }
302                 }
303             }
304         }
305     }
306 }
307
308 //Delete player Lasers from vector if they go off the screen or if they are no longer displayed
309 for (auto LaserIter = PlayerLasersVector.begin(); LaserIter != PlayerLasersVector.end(); )
310 {
311     if ((LaserIter->display == false) || (LaserIter->position.row == 1))
312     {
313         LaserIter = PlayerLasersVector.erase(LaserIter);
314     }
315 }
```

Fig 4. team35-terminalinvaders.cpp. Final iteration of the game. Builds on the chrono, terminal, and key listener functions in fishies.cpp. Core game logic was split up into individual functions to make it more modular.