Combat Robot

Data Structures's L2AC's Group 5's Final Project Report

Arvin Yuwono - 2502009721

Bayu Hartho Leksono – 2502013731

Darryl Maulana Hilmy – 2502009652

TABLE OF CONTENTS

**Problem Description**

   This project is a crude adaptation of Hero of Robots, a Taiwanese arcade trading game. It simulates where upon starting the game, the user acquires a new card which allows them to play with it or any other cards the user owns against enemies. This iteration is the best possible impression coded in C++, and the prototype has been proven to be partially successful, with some subsequent errors occurring occasionally.

**Data Structures Analysis**

- Vector

   The ParseCommaDelimitedString function separates a string via commas and places the separated strings into a vector. Using vectors as the data structure in this scenario seems to be the common practice in C++.

   First, A vector's size is dynamic, allowing worry-free amounts of insertion and deletion, unlike arrays, whose sizes are fixed. Second, when compared to List, the space of each element in a vector is only half of an element in a List because List traverses using nodes. And since the insertion of the substring is done at the end of the data structure, both Vector and List have the same cost in terms of time. Thus, in overall performance, vector is the better choice. Vectors have a random access iterator, allowing the values to be accessed by index and then used to create objects of the type Robot that are later on ushered into the vector. Using an iterator would be too bothersome. A vector of the type Robot is initialized with the aforementioned function, which is later accessed by a random index.

```
// Seperate a string by commas and put the seperated strings into vector
vector<string> parseCommaDelimitedString(string line){
vector<string> result;
stringstream s_stream(line);
while(s_stream.good()){
string substr;
getline(s_stream,substr,',');
result.push_back(substr);
}
return result;
```

- Queue

    The data structure that is often used to store game objects are array, queue and list. Here in our game, we want the data structure to be able to represent the order of the enemies encountered by the player and are placed in a queue to ramp up the difficulty of the simulation. It seems to make much more logical sense with 'First In First Out', where the first enemy to be faced and defeated is removed first. Using a queue, removing the element at the front will only cost O(1) time, which may be of significant difference in large amounts of data, when compared to List, which costs O(n) time.

    The table below provides an insight on the execution time for the enemies setup, which covers the following: 3x push, 1x front, and 1x pop. According to the result, there is really not much difference, which may be due to the small amount of data. However, it can still be seen that the queue is the winner.

|  | Queue | List |
| --- | --- | --- |
| Best Time | 0.57 ms | 0.54 ms |
| Average Time | 1.03 ms | 1.05 ms |
| Worst Time | 1.57 ms | 1.66 ms |

Table 1.1 Execution Time of Enemies Initial Setup

```
298        enemies.push(e1);enemies.push(e2);enemies.push(e3);
299            currentEnemy = enemies.front(); //Challenge the first one in the queue
300            enemies.pop(); //To prevent being called again
```

Image 2.1 Queue Insertion and Deletion

```
294    //        enemyList.push_back(e1); enemyList.push_back(e2); enemyList.push_back(e3);
295    //        currentEnemy = enemyList.front();
296    //        enemyList.pop_front();
```

Image 2.2 List Insertion and Deletion

- Map

    A Dictionary-Esque data structure that prevents the occurrences of identical keys. Not only would it be used for differentiating between robot names, but reducing the amount of iteration as well. Unlike vectors, Map will have an easier time to find specific keys. With other data structures using a type of class, one would have to use the getter method during each iteration to check for confirmation. Unfortunately, C++ Builder does not seem to support .contains() function, which was added in C++ 20. Thus, in our case, the find algorithm was used to check whether the key existed in the text file in the first place to prevent unwanted errors.

```cpp
void __fastcall TQuestForm::StartPanelClick(TObject *Sender)
{
    map<AnsiString,vector<int>> newCardMap = GenerateCard();
    map<AnsiString,vector<int>>:: iterator newCardIt = newCardMap.begin(); // Iterate through all elements in map
    AnsiString newCardName;
    int newCardHealth,newCardMelee,newCardShooting,newCardDefense;
    while(newCardIt!=newCardMap.end()){
        newCardName = newCardIt->first; //Retrieve key
        vector<int> newCardValue = newCardIt->second; //Retrieve value
        //Set object value
        newCardHealth = newCardValue.at(0);
        newCardMelee = newCardValue.at(1);
        newCardShooting = newCardValue.at(2);
        newCardDefense = newCardValue.at(3);
        newCardIt++;
    }
    //Display the status information

    CardNameLabel->Text=newCardName;
    CardHealthLabel->Text=newCardHealth;
    CardMeleeLabel->Text=newCardHealth;
    CardShootingLabel->Text=newCardShooting;
    CardDefenseLabel->Text=newCardDefense;
    //Save the card in 'inventory' text file
    fstream cardDB;
    cardDB.open("cards.txt",ios::app);

    if(cardDB.is_open()){ //Write the data to the text file
        cardDB<<newCardName<<","<<newCardHealth<<","<<newCardMelee<<","<<newCardShooting<<","<<newCardDefense<<"\n";
        cardDB.close();
    }
    //Load all owned cards to map
    cardMap = LoadCards();
    //Switch the panel screen being displayed to card details
    StartPanel->Visible=false;
    CardPanel->Visible=true;
```

Image 3.1 Map Iteration

```cpp
void __fastcall TQuestForm:: StartRobot(){
// auto start = std::chrono::high_resolution_clock::now();
//std::this_thread::sleep_for(std::chrono::seconds (3));
        AnsiString rawName= CardEdit->Text; //Get user input from editable text field

        cardIt = cardMap.find(rawName);
        //Find reference: https://thispointer.com/how-check-if-a-given-key-exists-in

         // Find the element with key [cardName]
        // Check whether it exists in the cardMap or not
        if(cardIt!= cardMap.end()){
        //Element is found
        vector<int> cardValue = cardMap.at(rawName);  //Access the value from iterat
    //Retrieve the data
        int health = cardValue[0];
        int melee = cardValue[1];
        int shooting = cardValue[2];
        int defense = cardValue[3];
        //Set player's status value
        player.setName(rawName);
        player.setHealth(health);
    player.setMaxHealth(health);
        player.setMAtk(melee);
        player.setSAtk(shooting);
        player.setDefense(defense);
        //Display the status on text labels
    PlayerNameLabel->Text = player.getName();
        PlayerHealthLabel->Text = player.getHealth();
        PlayerMATKLabel->Text = player.getMAtk();
        PlayerSATKLabel->Text = player.getSAtk();
        PlayerMeleeDefenseLabel->Text = player.getDefense();
    PlayerRangeDefenseLabel->Text = player.getDefense();
    //Change displayed panel screen to Combat
        SetupPanel->Visible=false;
        CombatPanel->Visible=true;
        }
```

Image 3.2 Map Find Algorithm

**Game Description**

The game initiates with the player starting at the login interface, where the player logs into the game. The game will then transition with the player placed into battle and facing waves of AI enemies. The player will be granted an interface and will be able to select between a melee attack or a ranged attack. The enemy will initiate self-defense via an anti-melee or an anti-range. The enemy uses random number generation to determine which defense they will do.

Upon being executed, the decisions made by the player and the enemy will result in numerous cycles of attacking and defending. If the player achieves victory, they will transition to the next stage and recover all lost health points. If the player loses then the game will conclude.

**Program manual**

The .exe file is available in the Win32/Debug folder. The code/script responsible for the program can be found in the 'Quest' and 'Project1' files.
Step 1: Open the 'Win32' Folder

| Name | Date modified | Type | Size |
|---|---|---|---|
| __astcache | 13/06/2022 20.03 | File folder | |
| Game_Asset | 13/06/2022 23.26 | File folder | |
| Win32 | 13/06/2022 20.00 | File folder | |
| .gitattributes | 13/06/2022 20.00 | GITATTRIBUTES File | 1 KB |
| Account.cpp | 13/06/2022 20.00 | C++ Source | 3 KB |
| Account.fmx | 13/06/2022 20.00 | FireMonkey Form | 4 KB |
| Account.h | 13/06/2022 20.00 | C/C++ Header | 2 KB |
| Account.vlb | 13/06/2022 20.00 | VLB File | 1 KB |
| Lose.cpp | 13/06/2022 22.49 | C++ Source | 1 KB |
| Lose.fmx | 13/06/2022 22.50 | FireMonkey Form | 1 KB |
| Lose.h | 13/06/2022 22.50 | C/C++ Header | 1 KB |
| Project1.cbproj | 13/06/2022 23.35 | BCB Project File | 56 KB |
| Project1.cbproj.local | 13/06/2022 23.35 | LOCAL File | 7 KB |
| Project1.cpp | 13/06/2022 22.57 | C++ Source | 1 KB |
| Project1.res | 13/06/2022 23.35 | Compiled Resource Script | 113 KB |
| Project1PCH1.h | 13/06/2022 20.00 | C/C++ Header | 1 KB |
| Quest.cpp | 13/06/2022 23.37 | C++ Source | 20 KB |
| Quest.fmx | 13/06/2022 23.38 | FireMonkey Form | 273 KB |
| Quest.h | 13/06/2022 23.34 | C/C++ Header | 4 KB |
| Quest.vlb | 13/06/2022 23.38 | VLB File | 2 KB |
| README.md | 13/06/2022 23.47 | MD File | 1 KB |
| Robot.cpp | 13/06/2022 20.00 | C++ Source | 1 KB |
| Robot.h | 13/06/2022 20.00 | C/C++ Header | 1 KB |
| Win.cpp | 13/06/2022 22.49 | C++ Source | 1 KB |
| Win.fmx | 13/06/2022 22.49 | FireMonkey Form | 1 KB |
| Win.h | 13/06/2022 22.49 | C/C++ Header | 1 KB |

Step 2: Open the 'Debug' Folder

| Name | Date modified | Type | Size |
|---|---|---|---|
| Debug | 13/06/2022 23.38 | File folder | |

Step 3: Open Project1.exe

| Name | Date modified | Type | Size |
|------|---------------|------|------|
| Account.obj | 13/06/2022 20.00 | 3D Object | 989 KB |
| cards.txt | 13/06/2022 23.38 | Text Document | 2 KB |
| CharacterForm.obj | 13/06/2022 20.00 | 3D Object | 566 KB |
| Classes.obj | 13/06/2022 20.00 | 3D Object | 41 KB |
| GameOver.obj | 13/06/2022 20.00 | 3D Object | 538 KB |
| LoginForm.obj | 13/06/2022 20.00 | 3D Object | 978 KB |
| MainForm.obj | 13/06/2022 20.00 | 3D Object | 628 KB |
| Menu.obj | 13/06/2022 20.00 | 3D Object | 538 KB |
| NumberWizard.obj | 13/06/2022 20.00 | 3D Object | 631 KB |
| Project1.exe | 13/06/2022 23.38 | Application | 600 KB |
| Project1.ilc | 13/06/2022 23.38 | ILC File | 1.472 KB |
| Project1.ild | 13/06/2022 23.38 | ILD File | 192 KB |
| Project1.ilf | 13/06/2022 23.38 | ILF File | 5.632 KB |
| Project1.ils | 13/06/2022 23.38 | ILS File | 18.560 KB |
| Project1.map | 13/06/2022 23.38 | Linker Address Map | 1 KB |
| Project1.obj | 13/06/2022 23.09 | 3D Object | 525 KB |
| Project1.pdi | 13/06/2022 23.38 | PDI File | 1 KB |
| Project1.tds | 13/06/2022 23.38 | TDS File | 13.248 KB |
| Project1PCH1.pch | 13/06/2022 20.00 | Precompiled Header File | 30.297 KB |
| Quest.obj | 13/06/2022 23.38 | 3D Object | 1.566 KB |
| RegistrationForm.obj | 13/06/2022 20.00 | 3D Object | 848 KB |
| robots.txt | 13/06/2022 20.00 | Text Document | 1 KB |
| Unit3.obj | 13/06/2022 20.00 | 3D Object | 538 KB |

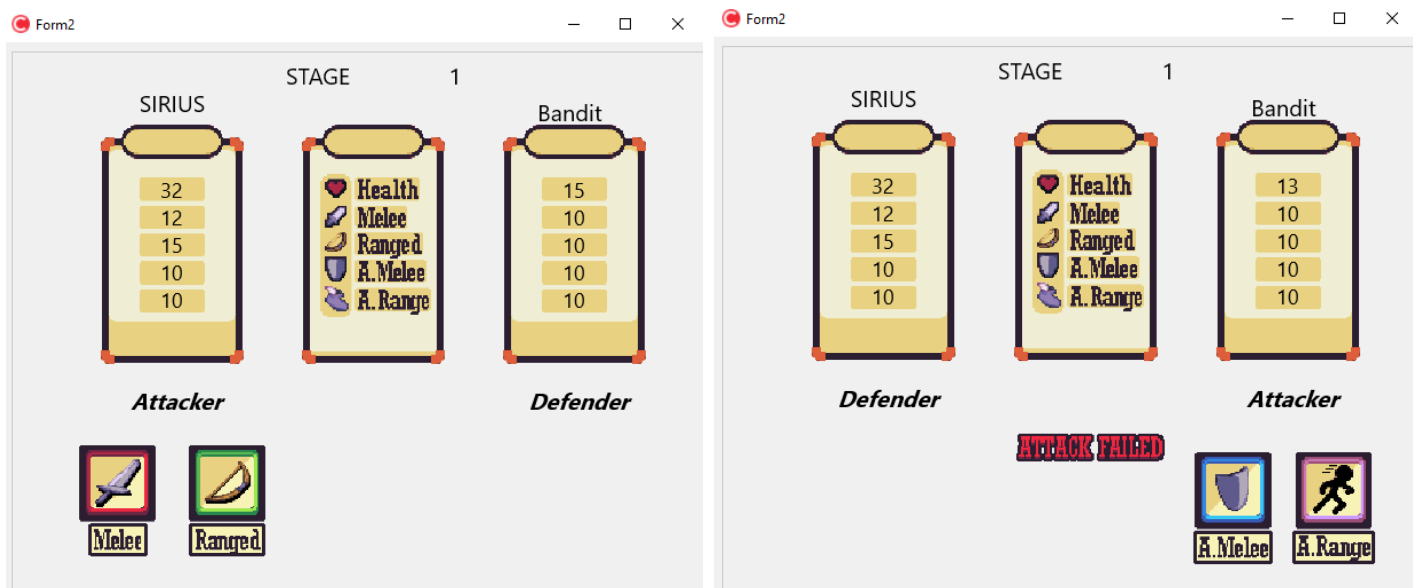**Execution of The Program**

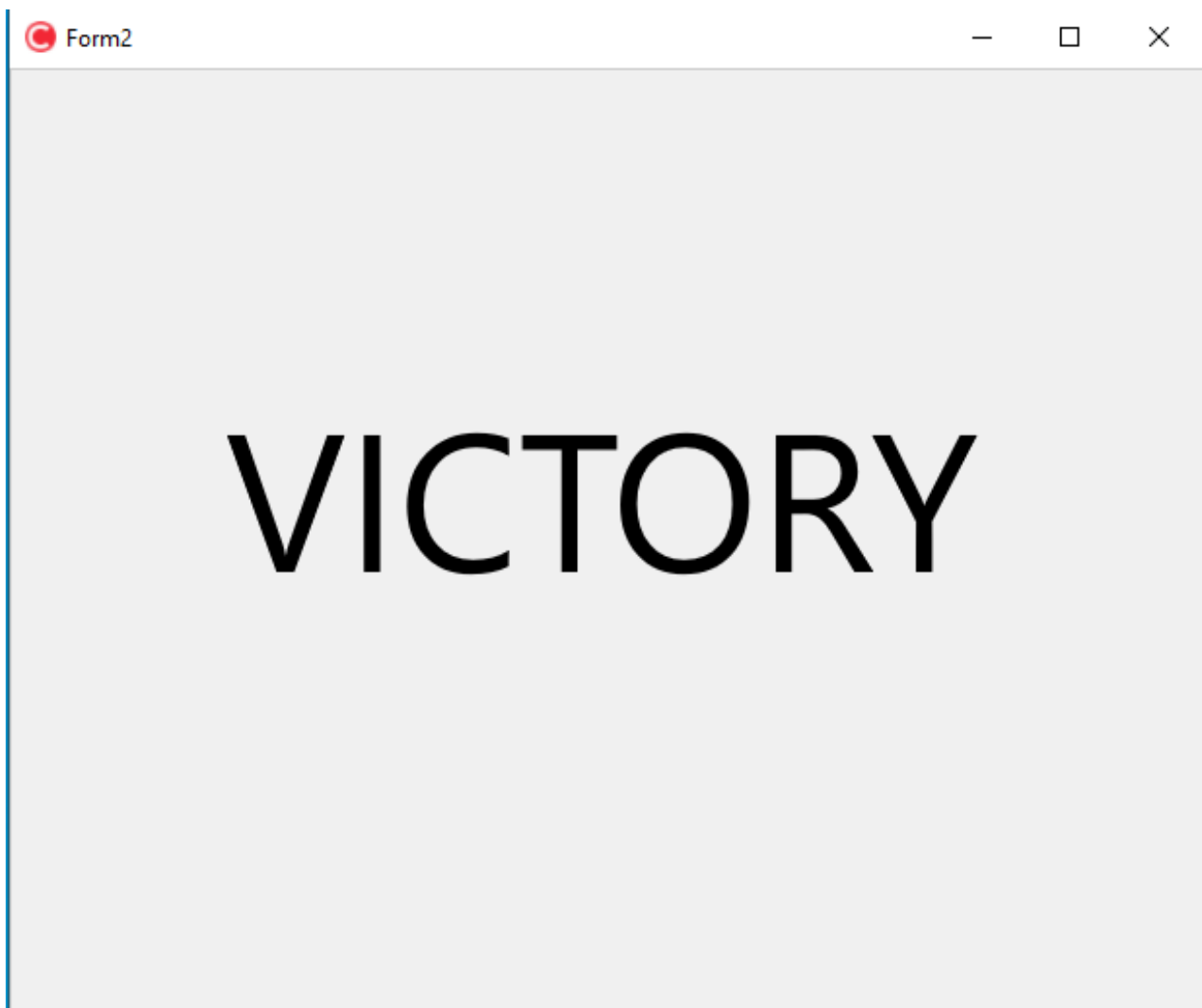Step 1: Receive a random robot card.
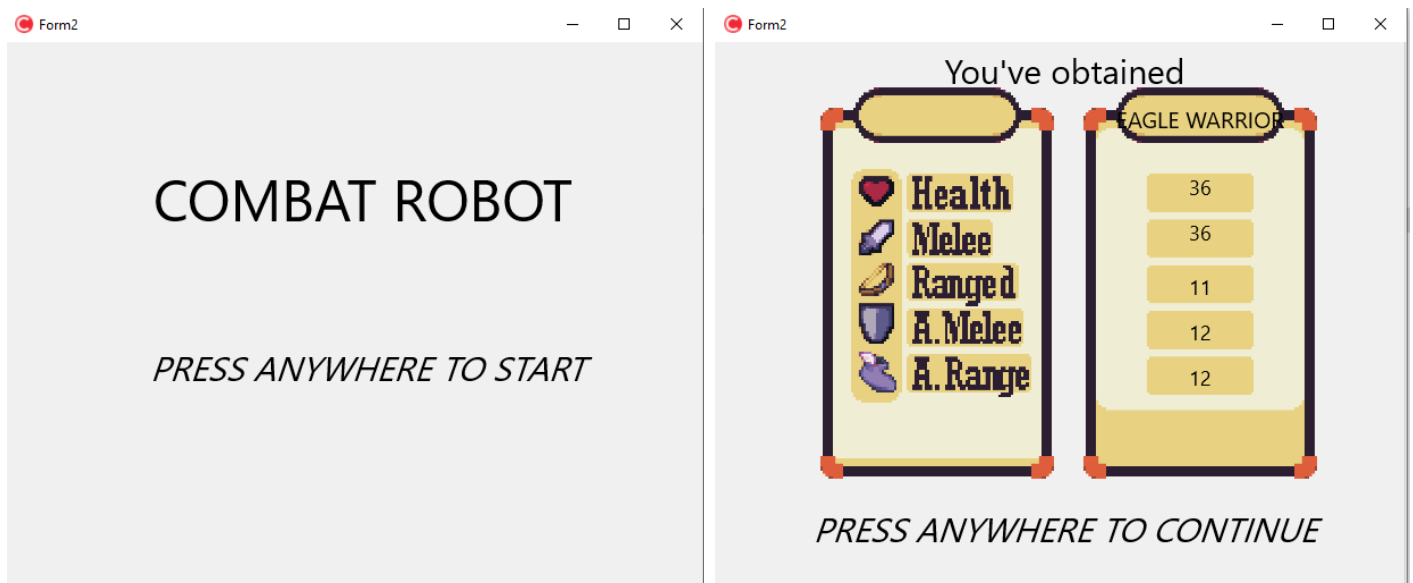


Step 2: Enter owned Card name.
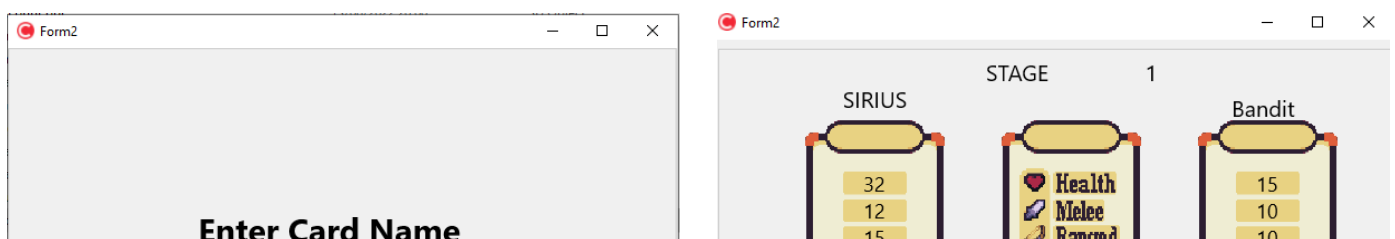
Step 3: Try to win in a combat system that depends on RNG.



Step 4: Either win or lose.

**Resources**

https://www.youtube.com/watch?v=nxHnnQToy5o&list=PL43pGnjiVwgQakzRxpt2amqN9f7-tRtc_&index=3

https://www.youtube.com/watch?v=EGCuStJyuVE&list=PL43pGnjiVwgQakzRxpt2amqN9f7-tRtc_&index=2

https://www.youtube.com/watch?v=UJI_SdxAtzk

https://thispointer.com/how-check-if-a-given-key-exists-in-a-map-c/

https://www.geeksforgeeks.org/difference-between-vector-and-list/

https://stackoverflow.com/questions/2074970/stack-and-queue-why

https://www.delftstack.com/howto/cpp/cpp-timing/#use-stdchronohigh_resolution_clocknow-and-stdchronoduration_cast-to-measure-execution-time-of-a-function