|  |
| --- |
| University of Detroit Mercy, College of Engineering and Science Department of Computer Science CSSE-5930 Graduate Design Project |
| Missile Strike |
| “Enemy plane destroyed!” |
|  |
| **Ravi Gowda** |
| **11/1/2015** |

|  |
| --- |
|  |

***Abstract***

***Missile Strike is a computer game where the player must eliminate all aircrafts in order to win. There are three difficulty levels: easy, medium, and hard. Each rising difficulty requires greater accuracy to win. The game is designed as a board game and is made up of an environment, tokens, markers, and randomizers. Tokens are placed onto the environment and markers indicate how well or how bad the player is currently doing. Randomizers make the game spontaneous and unstructured. With an effective game design comes and effective strategy. Missile Strike incorporates goals, planned methods for achieving these goals, impacts of missed attempts and the varying importance of pieces. Players must use strategy to achieve the goal, but must also be aware of the level requirements. All pieces have varying level of importance and need to be distinguished for the player to effectively be strategic. Along with the design and strategy is the mathematical foundation of the game. This is how we instruct the machine to carry out limitations in the game. Algorithms are composed to deal with certain limitations, we shall call constraints. The game has undergone many evaluations as it prepares to be launched into the public for testing.***

Introduction

Missile Strike is a game developed to be played by oneself. It imitates the board game “Battleship” because it is played by guessing coordinates on a map (board). These coordinates have aircrafts on them which must be eliminated in order to win. The player starts by choosing the difficulty for the game (normally there are 50 aircrafts generated if a difficulty isn’t set). Once a difficulty is chosen the game prompts the player with a message indicating the difficulty conditions. If the player satisfies the conditions the game prompts the player again displaying a winning message. Once the winning message is displayed the winner must start a new game or choose a difficulty again. To make the game more exciting a radar is built in which shows the player how many aircrafts are located in the designated row and column he or she selects. The radar also indicates how many attempts were used and how many aircrafts were hit. These helpful hints along with the difficulties make the game fun and challenging. Easy mode has 10 aircrafts and must be beaten in 50 attempts. Medium mode has 10 aircrafts and must be beaten in 40 attempts. Hard mode has 10 aircrafts and must be beaten in 30 attempts, proving to be a challenge as 10/30 attempts must be a hit. With these numbers: easy mode requires a 20% accuracy, medium mode requires a 25% accuracy and hard mode requires 33.3% accuracy. These percentages indicate an increasing amount of skill to accomplish with each rising difficulty.

The game is arranged on a 10 by 10 board known as a map marking each coordinate with an icon. The game starts off marking each coordinate unknown. When the player inputs a row and column and clicks strike, the map changes. The system checks whether an aircraft is located on the coordinate and changes the icon on that coordinate to a miss or hit icon; the hit icon indicating that an aircraft has been hit and the miss icon indicating that no aircraft is currently in that coordinate. A missile sound goes off once the player clicks strike. When an aircraft is hit an explosion sound goes off. These are audio cues to make the game enjoyable. When launching the game the aircrafts are randomly generated. Whenever a difficulty option, or new game is selected the aircrafts are randomly generated again on the map. A statement is also included that checks whether two aircrafts share the same coordinate, and if so then the aircraft checked is randomized again. The randomization happens in a “do..while” loop so that the aircraft is not generated in another aircrafts coordinate again.

The game is run in a graphical user interface as opposed to the console. The GUI is a nice platform for the game because it can be revalidated and it is simple to use. The GUI is composed of panels for the map and for the input/radar. The map is generated in a grid layout. The grid layout is 10 by 10 and is placed on the center of a border layout. The border layout is the arrangement of the main frame and is a 5 direction layout with directions facing north, south, east, west, and center. To illustrate the layout of the game, here is what the main frame will look like.

Row

Columnn

Map

Bottom Panel

Theory of Game Design

To make the game exciting a lot of factors were involved with the game design. These factors can be broken down into: components, activity, decisions, luck, and victory [1]. Most games are defined by four types of components. These components are: environment, tokens, markers, and randomizers [1].

The environment is the platform upon which a game will be played on [1]. Missile Strike uses an abstract environment because it simulates an actual sky zone, which is not actually visible to the player. Tokens are the pieces placed onto the environment and are different than the actual environment because they are dynamic [1]. They can either enter play, leave play, move, or change [1]. In Missile Strike, these tokens are represented as aircrafts. They enter play by revealing the exact position that they are located, they leave play by starting a new game, and there locations change by randomization. “Markers are elements which exist outside of the environment and which effectively replace a tic upon a piece of paper” [1]. There are scoring markers, board markers, and so on [1]. In Missile Strike, markers are made for row and column. The row markers are indicated by an ascending value 1-10 in a horizontal fashion. The column markers are indicated the same way but in a vertical fashion. Randomizers are factors which make the game spontaneous and also a bit lucky [1]. In Missile Strike random aircrafts are generated whenever launching, whenever a difficulty is selected, or whenever a new game is chosen by the player. This keeps the player constantly challenged as the token are never in the same place and also a bit lucky if he or she happen to choose the right target.

Activity is the mechanics that define component interactions in a game [1]. They are the ways the player can interact with components [1]. Token activity can include: conflict, placement, and removal [1]. In Missile Strike we see token activity by conflict when a token is not located where the player strikes. This indicates a miss and is indicated to the player by a miss icon. Another potential conflict that could occur is if two aircrafts were generated on the same location. To avoid this potential conflict, checkers are made into the code that determine if this occurs and to keep randomizing if it does. Placement is another activity that occurs whenever the map is created. Placement occurs by creating the aircrafts so that they are placed on different locations of the map. Removal is another activity that only happens whenever the game is reset. Removal occurs when the map is recreated. This can happen whenever the player selects a new game, a new difficulty, or when launching the game.

Decisions make the game interesting because they offer the player a chance to use intelligence as well as offer choices for the player to make [1]. In Missile Strike the decisions ultimately occur from player activity. Whenever a player chooses a row and column to strike, a radar comes up informing the player how many possible aircrafts are in the selected row and column. Based on this information, a player can use logic to make decisions. Decisions are limited by constraints. With this game the constraints are the dimensions of the map. There are only 10 possible rows and 10 possible columns, so a player that strikes outside this margin will not have this attempt be shown on the board.

Luck is implemented in the game by randomness. Luck occurs whenever a player hits an aircraft without a logical decision. This makes the game exciting for the player as he or she will never know if they hit a target when initially striking. Luck doesn’t always happen, and that’s why it makes it an interesting concept. Most players will enjoy having luck on their side, so when it does happen it should motivate the player to keep playing.

Victory on the other hand is different than luck, but can use this factor in order to help the player achieve victory. Victory is the ultimate goal of the game and is defined by aircrafts hit. Markers indicate how many aircrafts were successfully hit and depending on the game mode this determines victory. A player can achieve victory several ways depending on the game mode, but just as a player is constrained by decisions, a players attempt to win the game is constrained by markers. If a player goes over the number of attempts requires he or she will lose. These obstacles are a good reason for why a player should strategize while playing the game.

Strategy of Game Design

Strategy is defined as “a plan of action intended to accomplish a specific goal” [2]. Strategy is intended to make the game a challenging obstacle that requires proper planning to get over. Strategy games can be characterized by having goals, plan to achieve those goals, minimizing impact, and unimportance of individual pieces as they relate to the whole [2]. In Missile Strike we exhibit strategy through decisions the player makes throughout the game.

The goal of Missile Strike is to eliminate all aircrafts on the board. To accomplish this task the player must enter values that will strike positions on the board. Once the goal is achieved the player is alerted displaying a winning message. The player’s criteria to achieve the goal is changed depending on the difficulty. When easy mode is selected the player has 50 attempts to eliminate all aircrafts. When medium mode is selected the player has 40 attempts to eliminate all aircrafts. When hard mode is selected the player only has 30 attempts to eliminate all aircrafts. The attempts marker increments whenever a player strikes. To meet the criteria and accomplish the goal, the player needs to plan out what rows and columns to strike most often.

The player can cultivate a plan from first striking. When the first attempt is made, a radar comes up showing all possible aircrafts in that row and column. A good player will keep track of this in his mind when choosing another location to strike. The player needs to also keep track of how many attempts he or she has. Once they go over the attempts mandated for the difficulty level they will lose the game. Proper planning helps guide the player in his or her conquest to win the game.

One thing to keep in mind while designing the game is the impact of missed strikes. The player should not feel overwhelmed or be in a negative position if missing a possible target. To minimize the impact of missed strikes, it will not count toward any criterion other than incrementing attempts. The player should still be able to win the game if missing possible targets, but it should be harder and require more planning. Too many misses will put the player in a negative position and may even lead to a loss.

Another strategic factor implemented into the game is declaring certain tokens as “unimportant”. These tokens that are unimportant are the miss and unknown pieces. These pieces are generated to inform the user that he or she has missed a target or hasn’t declared a strike on that location. These pieces are still very important to the development of the game because they are indicators, but they should not be valued in terms of strategy.

Mathematical Foundation

Complex algorithms have been formed for the aircraft creation, and radar. These algorithms are unique and carefully thought out. Aircraft creation is placed in a “for” loop. The “for” loop parameter initializes a variable to act as a counter for how many aircraft will be made. This variable is checked with the static variable “*numberOfAircrafts”*, which will vary depending on the game mode. If smaller it iterates what is in the “for” block. Inside the “for” block, aircrafts[variable][0] (row) and aircrafts[variable][1] (column) are set to a random integer. The 0 element determines how many aircrafts in row. The 1 element determines how many aircrafts in column. Also inside the “for” block is another “for” block which will check if any two aircrafts are equal to each other. The variable inside the parameter is to act as the last created aircraft. This variable is set to 0 and is checked against the current value of the outer “for” loop. When smaller (which will not happen unless one aircraft has already been made) an “if” statement is executed that checks whether the last aircraft created for row and column are equal to the new aircraft that was made. If both the row and column are equal to the last one then the new aircrafts are randomized once again. This process happens in a “do..while” loop to make sure an aircraft is not created in the same location again. An example of the code below illustrates how this works.

**if**( (aircrafts[aircraft][0] == aircrafts[last][0])&&(aircrafts[aircraft][1] == aircrafts[last][1]) )

**do**

{

aircrafts[aircraft][0]=random.nextInt(10);

aircrafts[aircraft][1]=random.nextInt(10);

}

**while**( (aircrafts[aircraft][0] == aircrafts[last][0])&&(aircrafts[aircraft][1] == aircrafts[last][1]) );

The radar is very simple, but effective. The way it works is it has a “for” loop which checks all possible rows and all possible columns, and if they are equal to the strike array for row and column a variable is incremented that counts how many aircrafts are in that row, and how many are in that column. For instance, if someone inputted 9 as the row to strike, the system will check that input to each possible aircraft in the row element for aircrafts. If there is a match, a variable named row will be incremented. This process works the same for column. The algorithm for this equation is stated below.

**for**(**int** **line**=0 ; line < *numberOfAircrafts* ; line++)

{

**if**(aircrafts[line][0]==strike[0])

row++;

**if**(aircrafts[line][1]==strike[1])

col++;

}

Conclusion

In order to make a successful game many variables must be considered. The theory for the game design should be thought out to make the game fun for the player. Strategy should be applied to make the game worth playing. A mathematical foundation needs to be devised in order to eliminate constraints within the game and to give it features that make it strategic. These elements in a game make it successful when launching into the public.

Missile Strike incorporates many factors into the game design which will help it compete in the gaming market. The game simulates a real board game and can compete with others of the same category. It is unique unlike any other game in its category because it has a simple interface and changing icons. It is also a game to be played by oneself eliminating the need to find others wanting to play at the same time.

The strategic factors develop the player’s ability to think critically as well as making the game fair and enjoyable. Without these factors a player can win or lose without any kind of thinking making it a dull game. Missile Strike constitutes as a strategy game as a planned method is required to win the game. A player with a good amount of strategy is more likely to beat the hardest difficulty then a player with little strategy.

The mathematical foundation is the basis of any game. We need to look at what conflicts may arise in the game and what to do about it. Missile Strike has constraints that need to be dealt with. The machine does not know that two aircrafts should not be assigned to the same location so we need to instruct it to avoid doing so. Other features need to be instructed as well because the machine cannot think. This is why composing algorithms for the code is the most important aspect of the game. It instructs the machine on how the game should play out.

References:

[1] "Gone Gaming." *: A Theory of Board Game Design: Definitions of Terms*. N.p., n.d. Web. 31 Oct. 2015.

[2] "The Strategy Game Designer's Constitution - Game Design - Articles - Articles." *GameDev.net*. N.p., n.d. Web. 01 Nov. 2015.