Game Report

# Escape

AIM: To get out of the building (entering the exit room), without being eaten by the monster.

Background information: The player is in the same building as a monster, in order to get out the player must navigate through the rooms to get to the exit room without running into the monster. The monster will move automatically towards the top left corner. The monster uses its senses to help detect the player. While navigating through the rooms the player has an opportunity to go through ability rooms, to give power ups/debuffs and to boost their score.

### Rules:

* Player starts in the top left corner. The monster starts in the bottom right corner.
* Player can move into any surrounding room diagonally or horizontally using the arrow keys.
* 1 point added to score every time the player moves into a room.
* For every move the player takes, the monster takes a move.
* The goal is to get to the exit room without running into the monster.
* If the player ends up in the same room as the monster, the game is over.
* Some rooms contain ability chests
* Ability chests contain a mixture of power ups and debuffs which are randomly generated when a player enters a room containing an ability chest.
  + **Freeze debuff –** Power down which causes the player to not move for three turns.
  + **Speed ability -** Power up which locks the monster in a room for 2 moves.
  + **Invisible debuff –** Power down which turns the monster invisible for 2 moves.

## Key Features

#### Main Menu

The main menu will be the first page the user will see when they start the game. The purpose is to show the user all the options they have within the game. The main menu contains 4 buttons; ‘Start’, ‘How To Play’, ‘High Score’ and ‘Back To Menu’.

#### Start Game

The start button will run all the functions to start the game, such as the player icon, monster icon, ability chests, exit icon and the grid. ’Back To Menu’ is a button which is disabled and not visible within the main menu.

#### How To Play

The ‘How To Play’ button will bring up instructions on how to play the game when it is pressed. When the user presses this button, the ‘Back To Menu’ button is enabled and becomes visible.

#### High Score

‘High Score’ displays the name and score of the user with the current highest score. The high score is saved in local storage so will only use the scores which have been earned on the specific browser. If no name is entered when the high score is made ‘Unknown’ will be used as the name. The ‘Back To Menu’ button is enabled and visible on this page to allow the user to return to the menu.

#### Back To Menu

‘Back To Menu’ is a button which starts off disabled and hidden in the main menu. The button is enabled and visible within the ‘How To Play’, ‘High Score’ section as well as in game. The purpose of the button is to return the user back to the menu, so it is only required within those sections of the game.

#### Player

When the game is loaded up, the player will start off in the top left corner, whereas the monster will start in the bottom right corner. The user will be able to navigate through the rooms using arrow keys, which will only register valid moves, if the user is in the top left corner, they will not be able to move left or up, as they are on the edge.

#### Ability Chest

If the player enters a room with an ability chest, a random ability will be selected out of ‘Speed Ability’, ‘Freeze Debuff’ or ‘Invisible Debuff’ and will be applied to the player. When a specific ability is selected, the ability name will appear above the grid within the canvas for half a second. If the ‘Freeze Debuff’ is selected, the user will have to attempt to move twice before they are allowed to move again. Once the ability has been activated, the ability chest is removed and an ability can no longer be obtained from the specified position. If the monster enters an ability room, it will eat the ability chest so that it cannot be used by the player, the monster does not obtain an ability.

#### Monster

The monster starts off in the bottom right corner. It will take the first move when the user tries to make a move and then after will move based off the users move. The AI of the monster is to look at all of the cells within the same column and row of the monster before it takes a move. If the monster does not spot the player, it will by default move to the top left corner. After the monster has taken a move it will check the cells horizontal and vertical to its new position. If the monster finds the player, it will move in its direction, for example if the player was above the monster its next move would be to move up. Even when the monster has spotted the player, it will keep checking all the cells in the row and column to see if the player has moved direction. By adding a check before the monster will be able to see exactly where the player is if the monster is a cell behind.

#### Scoring

Scoring will be shown above the grid and canvas (animation area), to the right of the back to menu button. The score is updated after each move the player has made. For each room the player enters +1 point is added to the score, if the player enters a room containing an ability chest +10 will be added to the score. If the player has a freeze debuff on them, they will not gain a point for attempting to move.

#### Exit Room

The exit room, is the room which contains a fire exit sign. If the player enters this room, the game ends and the player has won the game.

#### Game Won

If the game has been won, the game ends and the ability chest as well as the player icon is removed. The game stops the user from moving anymore. A message is displayed telling the user they have won the game in the canvas. The high score is retrieved from local storage, if the current score is higher than the high score, it is replaced and the user is prompted to enter in their name. If they have not beaten the high score, no prompt appears.

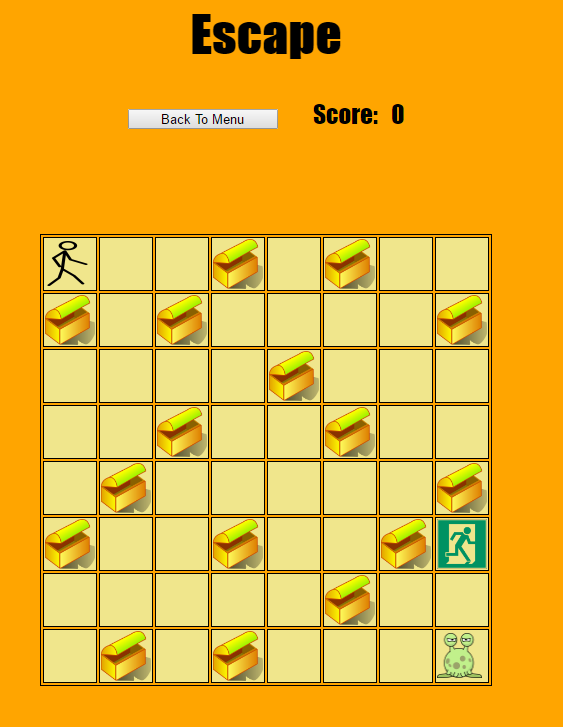
#### Game Lost

There are 2 ways in which the game can be lost; by entering the same room as the monster or meeting paths with the monster. For example if the player is in the room in the top left corner and the monster is directly below it, if the player then decides to enter the room below and the monster moves to the top left corner, they cross paths resulting in the game being lost. When the game is lost, a message will appear in the canvas telling the user that they have lost the game.

## Design

The first step towards the creation of the game was the grid. This essentially is the game map which the player interacts with. To create the grid an 8x8 table was originally created using HTML. Although a problem occurred when the main menu was created, as there was a difficulty hiding the table. In order for the table to be hidden, it had to be created in JavaScript. This involved creating new rows and cells; each cell was located a unique number as a cell id. The basis of this was that the cell ID would be used to map where each specific element within the game is. The player image was placed in cell number 0 at the start of the game and the monster was placed in cell number 63 at the beginning of the game.

Back To Menu button



Monster start position

Exit room

Score

Canvas which displays text

Ability Chest

Player start position

Figure 1 Game Grid

Keyboard detection is required to be able to detect specific arrow keys pressed within the game, this was done using switch statements as it is more efficient than if statements. When a specific arrow key is pressed the switch statement looks up the case to match it, e.g. when the left arrow is pressed, the variable ‘I’ is assigned the value 0, which is then passed into the arrowMovementFunction.

To make the game as interactive as possible, a main menu was created to allow users to see how to play and to display the current high score within the game. In order to have a functioning main menu, buttons were used which contained event handlers. For example the Start Button used event handlers to launch ‘loadFunction’ when pressed. The ‘loadFunction’ is a function which contains specific functions all designed to load the game. Many of the functions have specified values which are passed into it, such as playerFunction(0) passes the value 0 into it, which sets the value of variable x. Variable x is used to define where exactly the player icon will be within the grid.

PlayerFunction was a very difficult function to implement as this involved translating values into a position on the grid using the player image. Originally it simply moved the player icon from one cell to another, by deleting the image in the previous cell and then inserting the cell in the newly assigned image. Although as the game became more complex, features such as ability chests were implemented, if statements had to be used to check exactly where the player was on the grid and what to do in scenarios. If statements containing multiple checks were used such as ‘if (x == 5 && boolChest1 = true){‘, the && was used to give the if statement 2 conditions which needed to be met in order for it to run. The reasoning behind this was to make sure the player can only obtain an ability from each ability room once. If statements were also used in this function to check if the player and the monster were in the same room (game lost) as well as checking if the player is in the exit room (game won). Below displays a flow chart of the processing that occurs throughout the function.

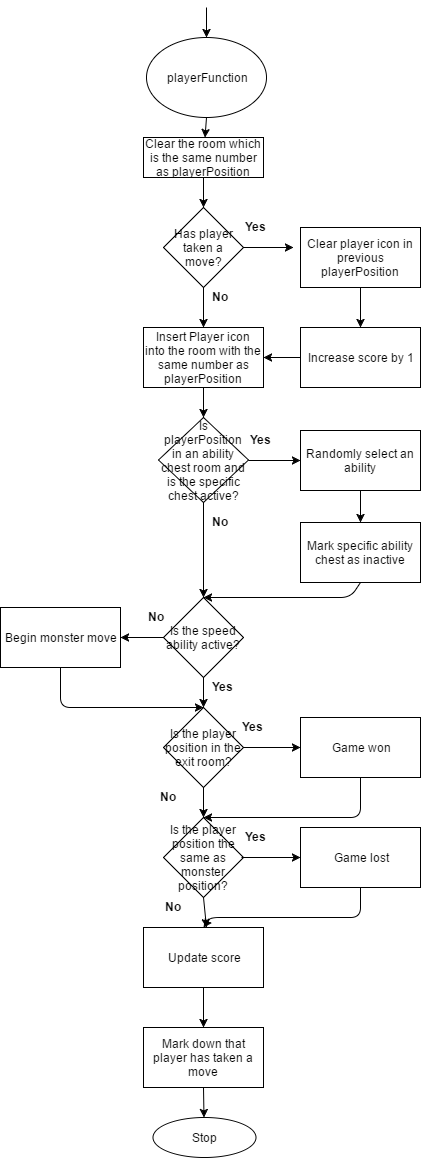


Figure 2 playerFunction flow chart

Allowing the player to move around the grid is no challenge for the user unless the monster is able to detect where the player is and react based on their move. To help implement this into my program an AI had to be created for the monster.

The monsterMovementFunction was the most difficult part of the program to create as an algorithm which can react to every single type of move the player makes had to be made. Originally when creating the pseudo code, it was designed so that the monster would search all the cells in the same column and row as the monster before it moved and would collect all the cell information in an array (see figure 8 in appendices for search function pseudo code). These arrays are then checked using a for loop within the checkCells function. The monsterMovementFunction passes the values of either the ‘rLArray’, ‘rUArray’, ‘rRArray’ or ‘rDArrary’ and playerPosition. The for loop measures the length of the array and then checks each individual element to see if it matches with the value of playerPosition. If a match is found, the function returns a true value, else it returns a false value. If statements were then used to check which array contained the playerPosition and if it was found to notify the monster to move accordingly (see figure 5 in appendices for check cells flow chart). For example if the ‘rRArray’ (which contains information about the cells to the right of the monster within the same row) has playerPosition within it, mark that player has been found and to move to the right through the use of Boolean variables (playerFound = true). Then an if statement is introduced to see if the player has been found in any of the arrays (if (playerFound == true)). If the player had not been found than a function named ‘defaultMove’ is run which gives a value to make the monster move towards the top left corner. If the player has been found, a nested if statement is run to see which direction the monster should move ‘else if (rightMove == true)’ and if this boolean is true, a value will be given to mark the monster to move right. The value is then passed into arrowMonsterMovement function which then maps where the monster is and creates the next monsterPosition. This monsterPosition is passed into the monsterFunction to move the image from the previous position towards the next position.

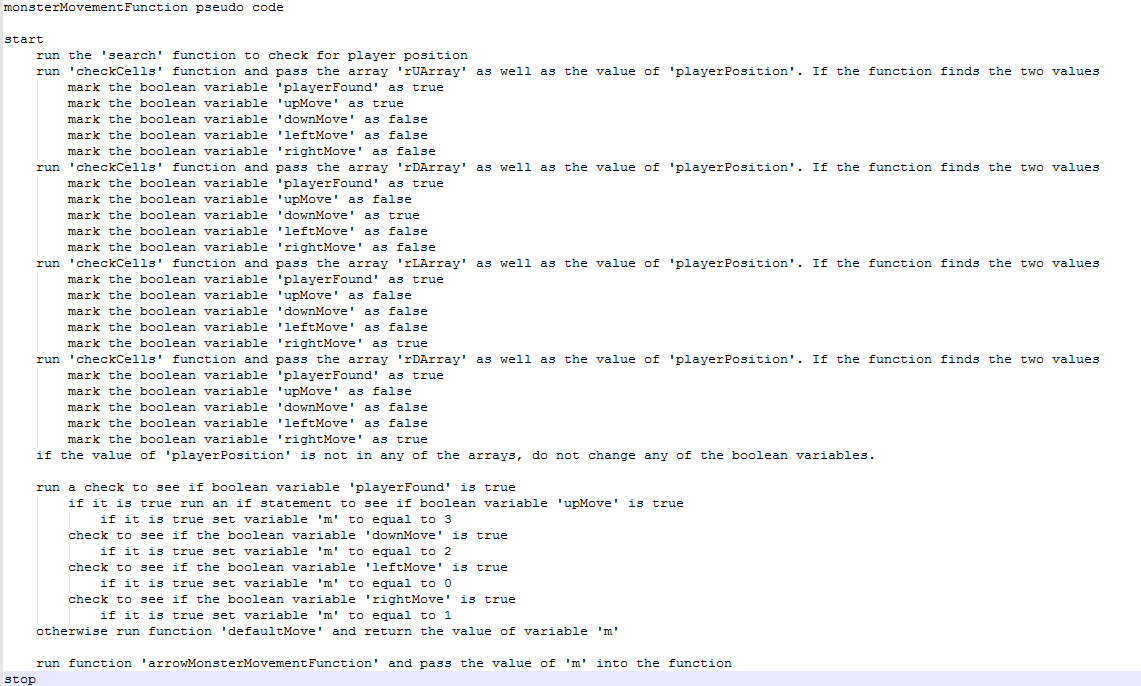


Figure 3 original monsterMovementFunction pseudo code

This function caused an imperfect monster AI as the monster was only checking player position before the move. It came to my attention that in order for the monster to keep track of the player accurately, the monster would have to check for player position before and after it takes the move. To help make the function correct, the if statements which were used to change the Booleans were put into 2 separate functions named ‘monsterCheckBefore’ and ‘monsterCheckAfter’. They both did identical if statement checks, the only difference was that they had separate variable names to distinguish the difference between the functions. The ‘monsterMovementFunction’ runs the ‘monsterCheckBefore’ function and returns Booleans. These Booleans are then checked using if statements and then mark which direction the monster should move, the ‘arrowMonsterMovementFunction’ is run as well as the ‘monsterCheckAfter’ function as a result. If the player has not been found then the Booleans from the monsterCheckAfter function are checked using else if statements, if any of the values are true, then the ‘arrowMonsterMovementFunction’ is run as well as the ‘monsterCheckAfter’ function. Else if the player has not been spotted the ‘defaultMove’ function is run with the ‘monsterCheckAfter’ and ‘arrowMonsterMovementFunction’. See figure 9 in appendices to see the altered pseudo code.

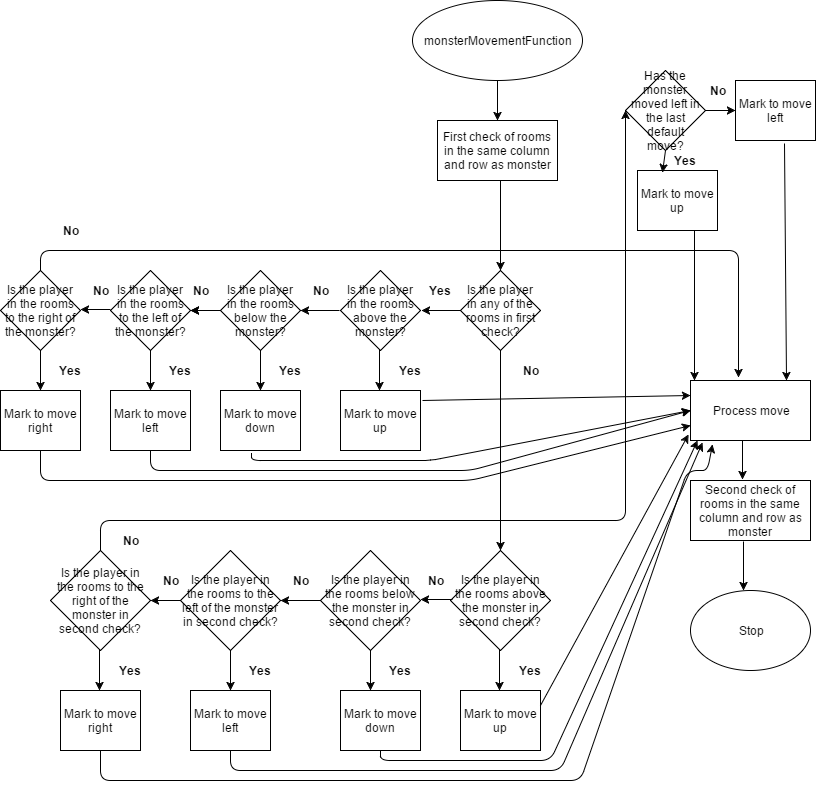


Figure 4 monsterMovementFunction flowchart

Another key feature of the game is the ability chests. In order to generate a random ability, a random number is selected between 1, 2 and 3 when the player enters an active ability chest room. This involved using ‘Math.random’ to help select a number and resulted in a specific ability function being run based on the number selected. When an ability function was run, accompanying text appeared which displayed the specific ability which the player has gained, which caused for text to stay until the player entered the next ability room. It took a lot of time to debug this problem and in the end, the solution used was a timer. A timer was introduced into each ability function so that after 0.5 seconds, the text was cleared (see figures 6 and 7 in appendices for further design of ability chests).

As mentioned previously, the playerFunction contains some if statements which check to see if the game has won or lost. If any of those statements run, most of the images within the grid will be deleted and the user will be notified that the game is over. In order to make sure that keyboard button presses are no longer being registered a gameOver Boolean was created, which only checks for keyboard presses while the Boolean is false. As soon as either the gameLostFunction or the gameWonFunction runs, this Boolean becomes true and the user can no longer move through the grid. The function also deletes the ability chests and player icon to show that the game is over. A highscore function was added which used if statements to compare the current score with the local score; this is retrieved from local storage and if the current score is higher, the user will be prompted to enter their name.

## Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No** | **Description** | **Planned Outcome** | **Result** | **Test pass/fail** | **Fix** |
| **1.** | Launch game | Game menu appears | Game menu appears | Pass | n/a |
| **2.** | Press ‘Start Game’ button | Game starts, loads the grid, player icon, monster icon, chest icons, score text box and back to menu button | Game starts, loads the grid, player icon, monster icon, chest icons, score text box and back to menu button | Pass | n/a |
| **3.** | Press ‘How To Play’ button | Instructions on how to play the game appear. Back to menu button appears. | Instructions on how to play the game appear. Back to menu button appears. | Pass | n/a |
| **4.** | Press ‘High score’ before game has been played | Text appears saying “No high score has been set” | Text appears saying “undefined” | Fail | If statement needed to be added which produces text if high score is null |
| **5.** | Press ‘High score’ before game has been played | Text appears saying “No high score has been set” | Text appears saying “No high score has been set” | Pass | n/a |
| **6.** | Press ‘Back To Menu’ button in High Score section | Navigates back to main menu | Navigates back to main menu | Pass | n/a |
| **7.** | Valid player move (press right arrow when game has loaded) | Player moves to the right cell. Score increases by 1 point.  Monster moves. | Player moves to the right cell. Score increases by 1 point.  Monster moves. | Pass | n/a |
| **8.** | Invalid player move (press left arrow when player is in the top left corner) | Player cannot move. Score increases by 1 point.  Monster moves. | Player cannot move. Score increases by 1 point.  Monster moves. | Pass | n/a |
| **9.** | Move player into ability chest room | Ability chest disappears. A randomly selected ability name appears in the canvas area. | Ability chest disappears. A randomly selected ability name appears in the canvas area. | Pass | n/a |
| **10.** | Freeze Debuff | Player cannot move for 3 turns. Monster moves with every turn attempt. Freeze Debuff appears in the canvas for 0.5 seconds | Player cannot move for 3 turns. Monster moves with every turn attempt. Freeze Debuff appears in the canvas for 0.5 seconds | Pass | n/a |
| **11.** | Speed Ability | Monster cannot move for 2 turns. Speed Ability text appears for 0.5 seconds | Monster cannot move for 2 turns. Speed Ability text appears for 0.5 seconds | Pass | n/a |
| **12.** | Invisible Debuff | Monster becomes invisible for 3 moves. Invisible debuff text appears for 0.5 seconds. | Monster becomes invisible for 2 moves. Invisible debuff text appears for 0.5 seconds. | Fail | darkCounter value changed from 1 to 0. |
| **13.** | Invisible Debuff | Monster becomes invisible for 3 moves. Invisible debuff text appears for 0.5 seconds | Monster becomes invisible for 3 moves. Invisible debuff text appears for 0.5 seconds | Pass | n/a |
| **14.** | Player above monster test | Monster moves up towards the player. | Monster moves up towards the player. | Pass | n/a |
| **15.** | Monster check before move test. | Player moves down from top right corner to the same row as the monster. Monster moves into the same room as the player. | Player moves down from top right corner to the same row as the monster. Monster moves to the row the player was previously in. | Fail | Added a check before the monster moves and a check after the monster moves. So that the monster does not lose track of the player. |
| **16.** | Monster check before move test. | Player moves down from top right corner to the same row as the monster. Monster moves into the same room as the player. | Player moves down from top right corner to the same row as the monster. Monster moves into the same room as the player. | Pass | n/a |
| **17.** | Monster moves to ability chest | Ability chest disappears, no ability activated. | Ability chest disappears, no ability activated. | Pass | n/a |
| **18.** | Player enters room where ability chest was previously | Player moves to room. No ability activated and no text appears. | Player moves to room. No ability activated and no text appears. | Pass | n/a |
| **19.** | Player moves into monster cell. | Game ends. Text appears stating the game is lost. | Monster moves into player past cell. Game continues. | Fail | Added a variable which records the last monster move. If player moves into this cell game ends. |
| **20.** | Player moves into monster cell. | Game ends. Text appears stating the game is lost. | Game ends. Text appears stating the game is lost. | Pass | n/a |
| **21.** | Player and monster are in the same cell. | Game ends. Text appears stating the game is lost. | Game ends. Text appears stating the game is lost. | Pass | n/a |
| **22.** | Player enters exit room. | Game ends. Text appears stating the game has been won. | Game ends. Text appears stating the game has been won. | Pass | n/a |
| **23.** | Win game with high score. | Game ends. Text appears stating the game has been won. Prompt message appears asking the user to enter in their name. | Game ends. Text appears stating the game has been won. Prompt message appears asking the user to enter in their name. | Pass | n/a |
| **24.** | Press ‘High Score’ button in main menu after a high score has been made. | High score name appears with the correct score. | High score name appears with the correct score. | Pass | n/a |
| **25.** | Refresh browser and check if high score appears in browser | High score name appears with the correct score. | High score name appears with the correct score. | Pass | n/a |

## How This Game Could Be Improved?

There were some features which were not implemented within the game due to lack of time. One of the features which was planned at the beginning was to have a high score feature which showed the top 10 scores in ascending order. This was difficult to implement and I struggled to find any resources which gave guidance on how to implement this without the use of a JSON library.

Another feature not implemented was random chest ability placement. Currently the game places the ability chests in the same rooms. Due to the way the program was coded, attempting to create random ability placement caused a lot of errors within the game and would have been very time consuming to correct.

The AI for the monster was coded correctly so that it always moves to the correct room dependent on the player position. The problem is that the monster spends the whole game chasing the player. To improve this game, a smarter AI could be implemented which can anticipate the players next move not just react to it. In order to be able to implement this more checks would be required, and some more complex equations would have to be introduced.

A final feature which could be added to give the game more longevity is a difficulty option before the game loads. When the user presses the ‘Start Game’ button an option of difficulties appears. Ranging from very easy, where the ability chests are less frequent and are more likely to select buffs rather than debuffs, to a very hard game where chests appear more often and are more likely to produce debuffs. This would require for a different number of chests to be placed as well as a randomizer which could have a bias towards debuffs or buffs.

## References

1. How to remove a html element - <http://stackoverflow.com/questions/5933157/how-to-remove-an-html-element-using-javascript>
2. How to disable a button - <http://stackoverflow.com/questions/17114825/how-to-disable-button-after-one-click-with-validation-using-javascript>
3. Adjust visibility of button - <https://www.w3schools.com/jsref/prop_style_visibility.asp>
4. Saving a high score to local storage - <http://stackoverflow.com/questions/29370017/adding-a-high-score-to-local-storage>
5. Local storage - <https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API#localStorage>
6. Local Storage - <https://www.w3schools.com/html/html5_webstorage.asp>
7. Create a HTML element - <https://www.w3schools.com/jsref/met_document_createelement.asp>
8. How to create spaces when creating a text node - <http://www.webdeveloper.com/forum/showthread.php?193107-RESOLVED-Insert-amp-nbsp-when-using-createTextNode()>
9. Adding an image in Javascript - <http://stackoverflow.com/questions/7802744/adding-an-img-element-to-a-div-with-javascript>
10. Using a timeout in Javascript - https://www.w3schools.com/js/js\_timing.asp
11. Clear a canvas - http://stackoverflow.com/questions/2142535/how-to-clear-the-canvas-for-redrawing
12. Detect keyboard press and handling the events - <https://www.kirupa.com/html5/keyboard_events_in_javascript.htm>
13. Switch and Case - <https://www.kirupa.com/html5/keyboard_events_in_javascript.htm>
14. Canvas - <https://www.w3schools.com/html/html5_canvas.asp>
15. Check for element in array - http://stackoverflow.com/questions/7378228/check-if-an-element-is-present-in-an-array

## Appendices

### 

Figure 5 checkCellsFunction flow chart

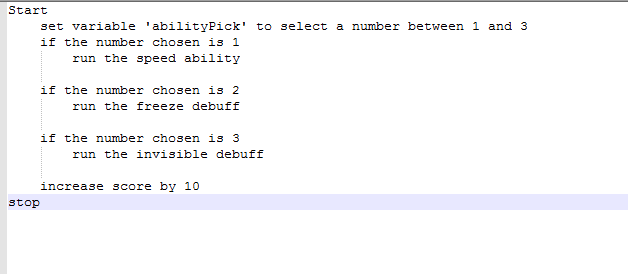


Figure 6 abilityFunction pseudo code

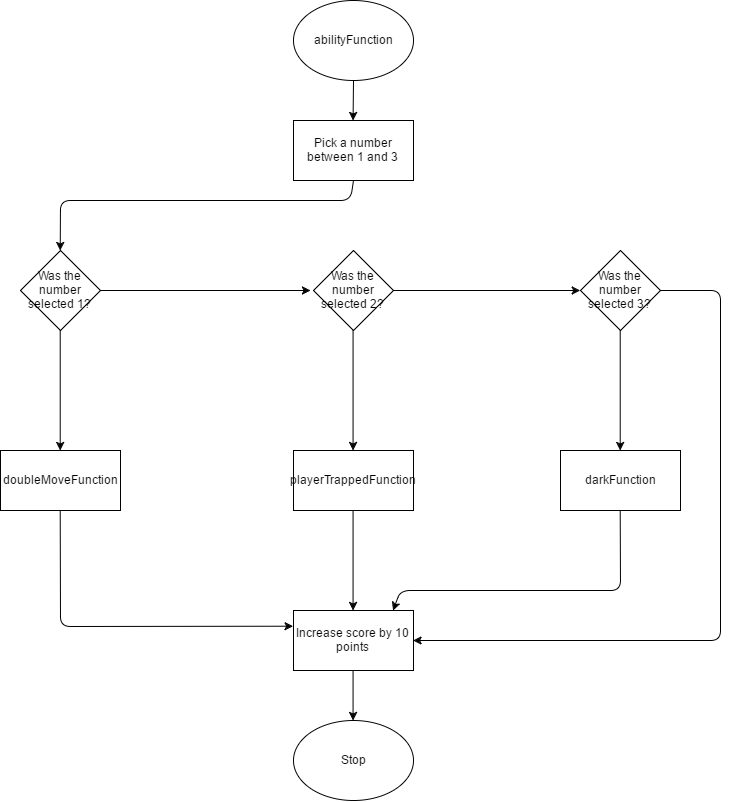


Figure 7 abilityFunction flowchart

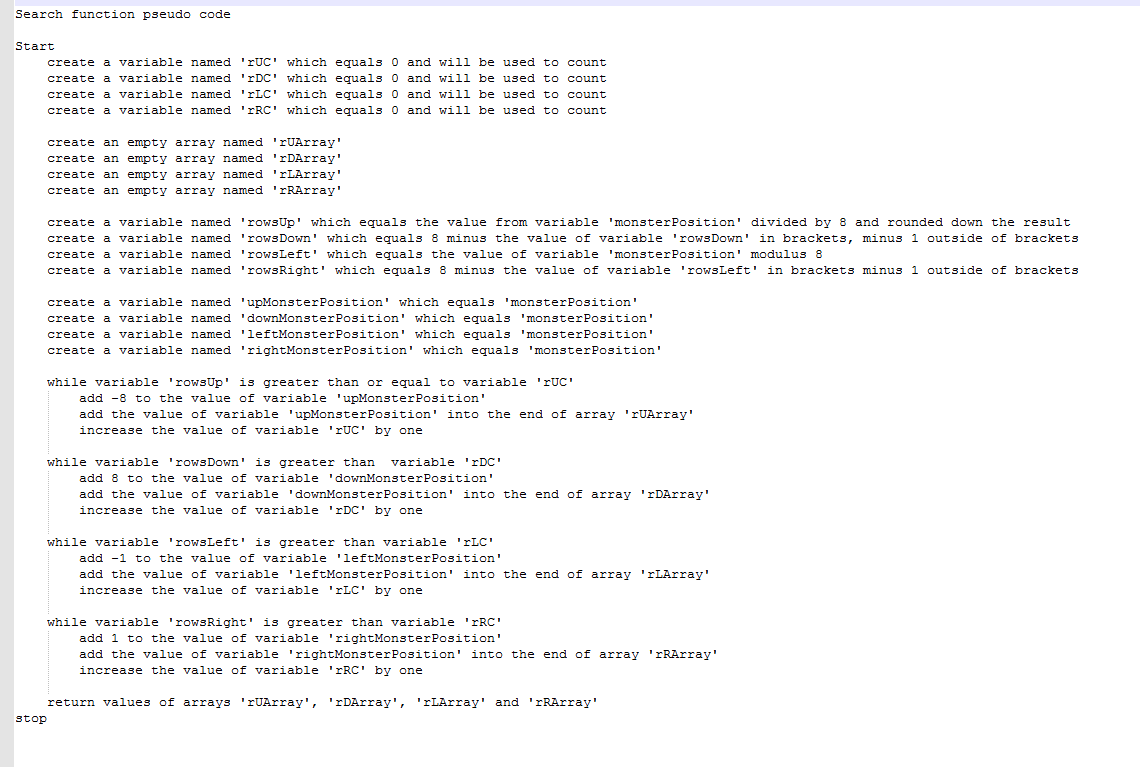


Figure 8 search function pseudo code

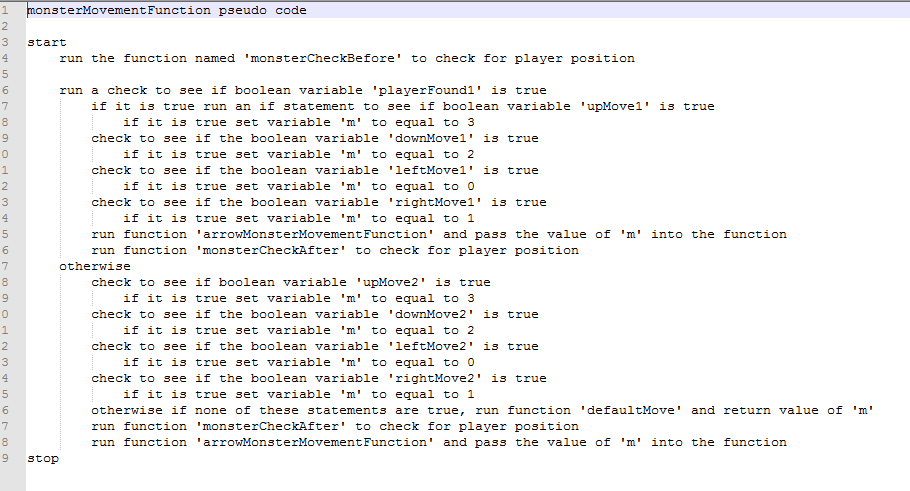


Figure 9 second version of monsterMovementfunction pseudo code