GCSE Computer Science

Gaming Scenario
Student 1 - Marty

User Requirements.

1) Game Outline:

- BotMod is a game where you guide a robot through a bird's eye landscape.
- The robot collects people, in order to carry them to a base camp located on the map.
- This objective has to be achieved before the robot's power runs out; otherwise the player will lose the game.
- The landscape is a 10x10 bird's eye grid, containing grassland, rocks and ice.

2) The Robot:

- The robot can only move 1 square at a time either up, down, left or right, therefore diagonal moves cannot be made, as well the fact that the robot is unable to move off-screen.
- When the robot moves to a square containing a person, then if the robot's passenger bay has sufficient space, then the person will be picked up; if not, then the person remains in the square.
- The robot always begins in the top-left square of the game, and starts with 150 power units.
- When the robot comes into contact with the base camp, all people in the passenger bay will automatically be unloaded.
- Before the game begins, the player is allowed to choose the type of traction used by the robot (wheels, tracks or skis) and the passenger bay size (small, medium or large). These will have implications on the use of power and movement of the robot [See "4) Modifications"].

3) The Terrain:

- The terrain is a 10x10 grid composed of a combination of grassland, ice and rock. Different traction types will run with varied amounts of power on these different terrains.
  - Grassland = Green
  - Rocks = Brown
  - Ice = White
- On the landscape, the objects are displayed as the following:
  - **R** = The Robot
  - **P** = A person
  - **B** = The Base

Name: Student 1 Gaming

Candidate Number: 0000

4) Modifications:

- Before the start of the game, the player has the ability to choose the robot's traction type and passenger bay size. The traction type can be any of the 3 listed below. This choice will have an impact on the power units that are used on specific terrain.
  - Wheels:
    - Grassland = 1 Power Unit
    - Rocks = 2 Power Units
    - Ice = 3 Power Units
  - Tracks:
    - Grassland = 3 Power Units
    - Rocks = 3 Power Units
    - Ice = 3 Power Units
  - Skis:
    - Grassland = 3 Power Units
    - Rocks = 3 Power Units
    - Ice = 1 Power Unit
- The second aspect of the robot that can be changed is the size of the passenger bay.
  - Large:
    - Maximum passengers at any one time = 3
    - Power cost = +2 units per move
  - Medium:
    - Maximum passengers at any one time = 2
    - Power cost = +1 unit per move
  - Small:
    - Maximum passengers at any one time = 1
    - Power cost = +0 units per move

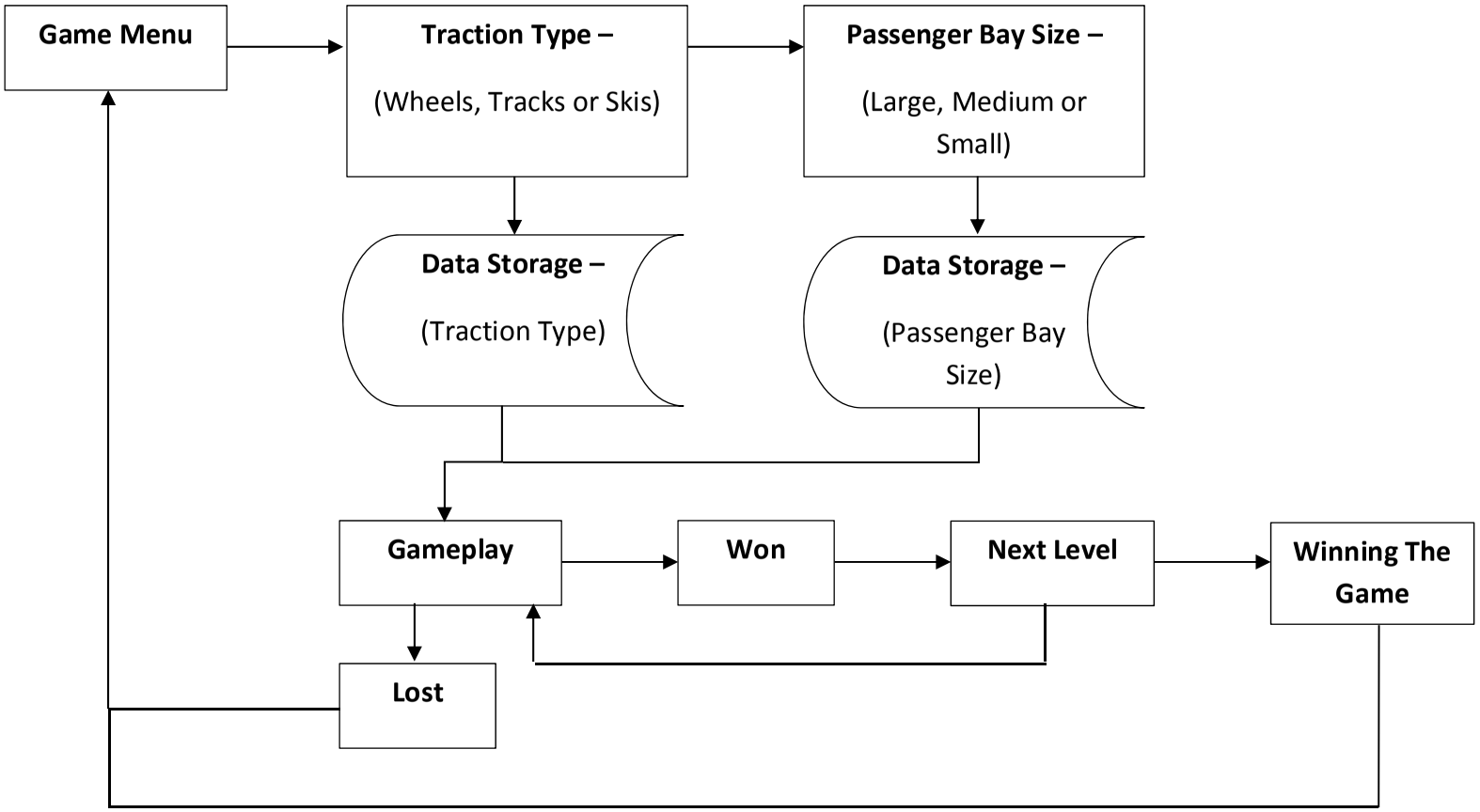
5) Level Progression:

- There is more than one level to the game. These levels will increase in difficulty and one of the factors causing this is the small decrease in power units.
- These levels will add substance and variation to the gameplay, perhaps introducing new elements into the mix.
- Other than slight additions, the robot, passengers and base remain, and function in the same way as before.

6) Winning & Losing:

- The game has an ending, where the user has successfully completed the game and has therefore won. They then have the ability to replay the game with different modifications on the robot.
- The game will also end when the robot runs out of power units, and therefore is unable to complete the current level. Again, the player is returned to the main menu as a result, giving them the ability to re-customise the robot, in order to attempt beating the game once more.

Game Overview.



Name: Student 1 Gaming

Candidate Number: 0000

BotMod Pseudocode.

```
FOREVER
IF background = "customise_robot_traction"
THEN
    FOREVER
    IF wheels_button SELECTED
    THEN SET traction_type = wheels
    ELSEIF tracks_button SELECTED
    THEN SET traction_type = tracks
    ELSEIF skis_button SELECTED
    THEN SET traction_type = skis
    ENDIF
IF background = "customise_robot_passenger_bay"
THEN
    FOREVER
    IF small_button SELECTED
    THEN SET passenger_bay_size = small
    IF medium_button SELECTED
    THEN SET passenger_bay_size = medium
    IF large_button SELECTED
    THEN SET passenger_bay_size = large
```

```
IF traction_type = wheels
THEN power_unit_effect = 1
ELSEIF traction_type = tracks
THEN power_unit_effect = 2
ELSEIF traction_type = skis
THEN power_unit_effect = 3
ENDIF
```

```
IF passenger_bay_size = small
THEN maximum_passengers = 1
ELSEIF passenger_bay_size = medium
THEN maximum_passengers = 2
ELSEIF passenger_bay_size = large
THEN maximum_passengers = 3
```

```
IF passengers_left = 0
THEN
    SHOW "level_complete"
    WAIT 1000ms
    SET level = level + 1
```

```
IF passengers_left = 0
THEN
    IF level = 5
    THEN SHOW "you_win"
    WAIT 1000ms
    SET level = 0
    SET main_menu = 1
    SET passengers_in_bay = 0
    SET base_passengers = 0
ENDIF
```

```
SET power_units = 150
SET level = 0
SET main_menu = 1
FOREVER
IF play_button SELECTED
THEN
    SET main_menu = 0
    SET level = 1
ELSEIF customise_robot SELECTED
THEN SET main_menu = 2
ELSEIF exit SELECTED
THEN STOP
ENDIF
```

```
FOREVER
IF main_menu = 1
THEN SET background = "main_menu"
ELSEIF main_menu = 2
THEN SET background = "customise_robot_traction"
ELSEIF main_menu = 2
THEN SET background = "customise_robot_passenger_bay"
ELSEIF main_menu = 0
THEN SET background = "level" + level
ENDIF
```

```
IF
    robot_x = passenger_x
    AND robot_y = passenger_y
THEN
    IF passenger_bay_size > passengers_in_bay
    THEN
        SET passsengers_left = passengers_left - 1
        HIDE passenger
        SET passengers_in_bay = passengers_in_bay + 1
    ENDIF
ENDIF
```

```
IF
    robot_x = base_x
    AND robot_y = base_y
THEN
    SET base_passengers = base_passengers + passengers_in_bay
    SET passengers_in_bay = 0
ENDIF
```

```
IF power_units = 0
THEN
    SHOW "game_over"
    WAIT 1000ms
    SET level = 0
    SET main_menu = 1
    SET passengers_in_bay = 0
    SET base_passengers = 0
ENDIF
```

```
IF terrain = grassland
THEN
    IF power_unit_effect = 1
    THEN SET power_units = power_units - 1
    ELSEIF power_unit_effect = 2
    THEN SET power_units = power_units - 3
    ELSEIF power_unit_effect = 3
    THEN SET power_units = power_units - 3
    ENDIF
ELSEIF terrain = rocks
THEN
    IF power_unit_effect = 1
    THEN SET power_units = power_units - 2
    ELSEIF power_unit_effect = 2
    THEN SET power_units = power_units - 3
    ELSEIF power_unit_effect = 3
    THEN SET power_units = power_units - 3
    ENDIF
ELSEIF terrain = ice
THEN
    IF power_unit_effect = 1
    THEN SET power_units = power_units - 3
    ELSEIF power_unit_effect = 2
    THEN SET power_units = power_units - 3
    ELSEIF power_unit_effect = 3
    THEN SET power_units = power_units - 1
    ENDIF
ENDIF
```

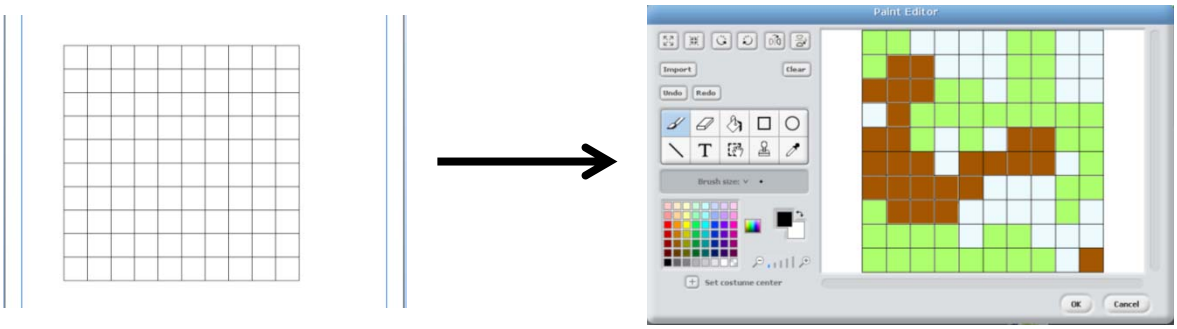
```
IF INPUT arrow_up
    THEN
        IF touching border = false
        THEN SET y = y + 36
        ENDIF
    ELSEIF INPUT arrow_left
    THEN
        IF touching border = false
        THEN  x = x - 36
        ENDIF
    ELSEIF INPUT arrow_right
    THEN
        IF touching border = false
        THEN  x = x + 36
        ENDIF
    ELSEIF INPUT arrow_down
    THEN
        IF touching border = false
        THEN SET  y = y - 36
        ENDIF
```

Name: Student 1 Gaming

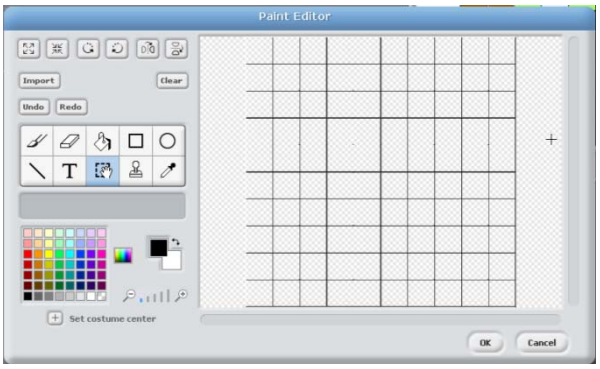
Candidate Number: 0000

Solution Development.

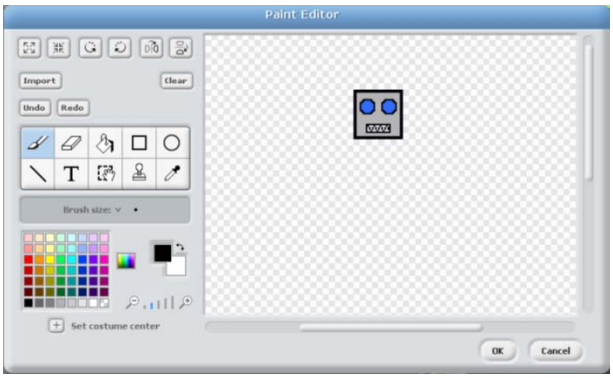
I used Serif DrawPlus to construct the basic background grid. After assigning the grid 10 rows and columns, I exported the grid as both a PNG and JPEG file separately, in case one format suited Scratch better than the other.



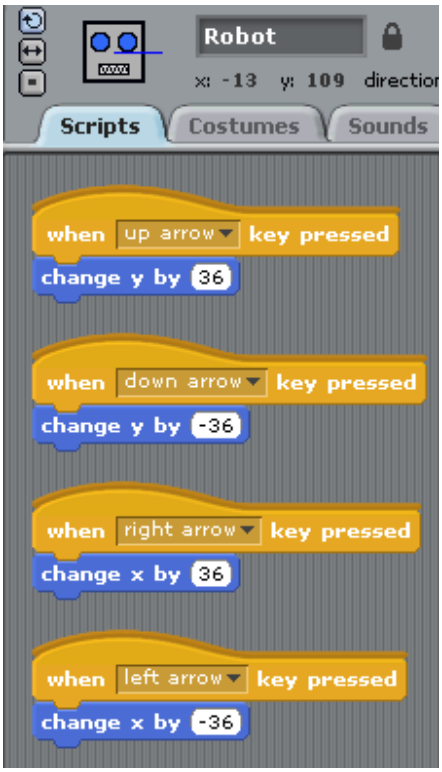
I also attempted to save the grid as an image from Word, in order to accurately resize the grid to scale with the Scratch stage before saving. However this didn't work and had occasional lines missing.



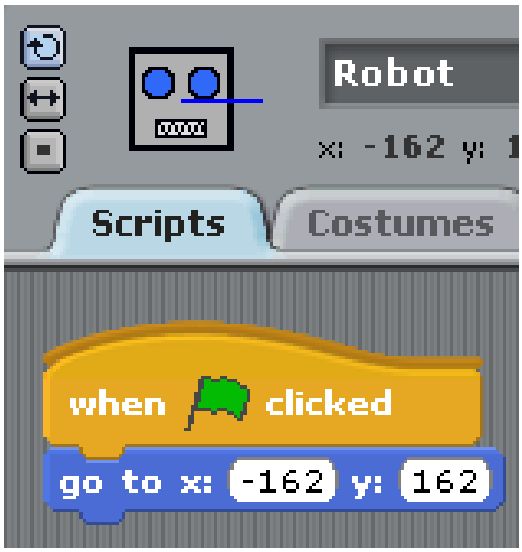
Instead of a simple letter 'R', I decided to construct a Robot sprite. At first the sprite was too big to fit in 1 square of the grid, but the end result was precisely 32x32 pixels.



I then moved on to the movement of the robot. Although I haven't yet added power units, I decided to make the robot able to move around the grid. At first I wanted to experiment with 'forever if' statements for motion, using 'sensing' commands to identify when an arrow key is pressed. I did this as in previous games, it allowed smoother motion, but I then realised that because the robot moves on a grid, it wasn't necessary and was less efficient.



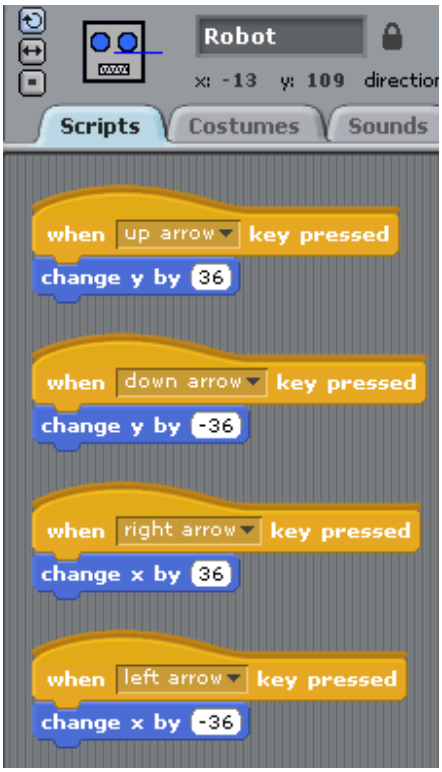
I then added a 'go to' command, which will return the robot to its starting position every time the game is started.



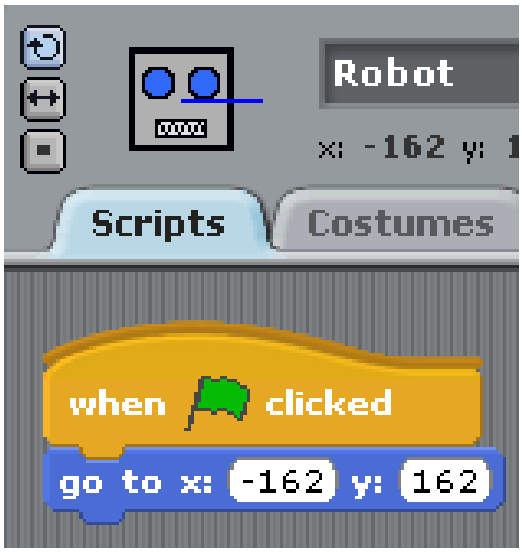
To prevent the robot from leaving the grid, I used an 'if' statement on all 4 arrow key movement commands.

I then modified the code, as the grid is ever so slightly not to scale, so that the robot moves 35.9 steps in any direction as oppose to 36. This made movement much more accurate within the grid, but I then needed to change the 'if' statement accordingly, by asking whether the position was greater than 161 or smaller than -161.

Next, I made a Main Menu sprite, with a title. The buttons would be added after I have constructed the appropriate variables.



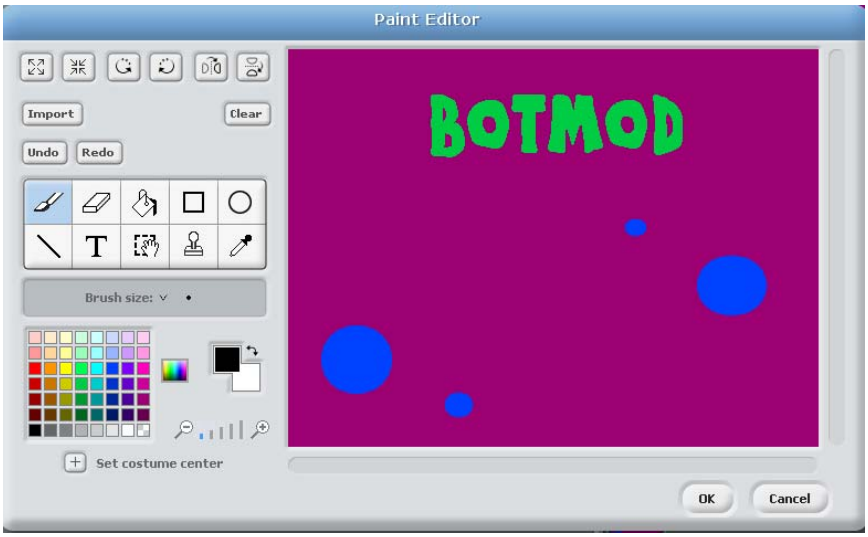
I then added a 'go to' command, which will return the robot to its starting position every time the game is started.



To prevent the robot from leaving the grid, I used an 'if' statement on all 4 arrow key movement commands.

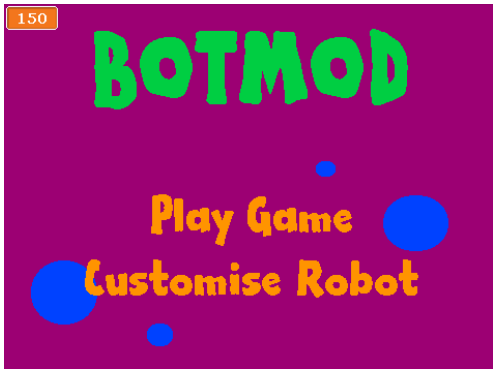
I then modified the code, as the grid is ever so slightly not to scale, so that the robot moves 35.9 steps in any direction as oppose to 36. This made movement much more accurate within the grid, but I then needed to change the 'if' statement accordingly, by asking whether the position was greater than 161 or smaller than -161.

Next, I made a Main Menu sprite, with a title. The buttons would be added after I have constructed the appropriate variables.



I then added both a 'power unit' and 'level' variable. The power unit variable I would deal with later, as I concentrated on the Main Menu. I used a forever command surrounding 2 separate 'if' statements both telling the Play Game sprite where to be on both the Main Menu and Customisation screens. When it is clicked, the game will start thanks to a broadcast, which the other sprites receive; the robot, people and base will all show at that point in their respective positions and the power unit variable is set to 150, to prevent the robot being moved prior to starting the game, which would reduce the power units without the game starting.

The next thing to do was add a button that takes the user to the Customisation menu, in which the traction and passenger bay size can be customised.

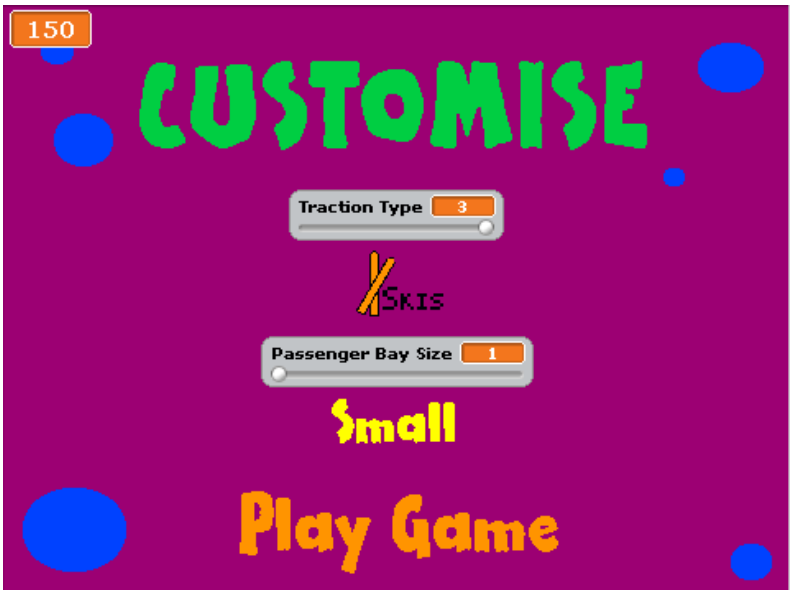


Name: Student 1 Gaming

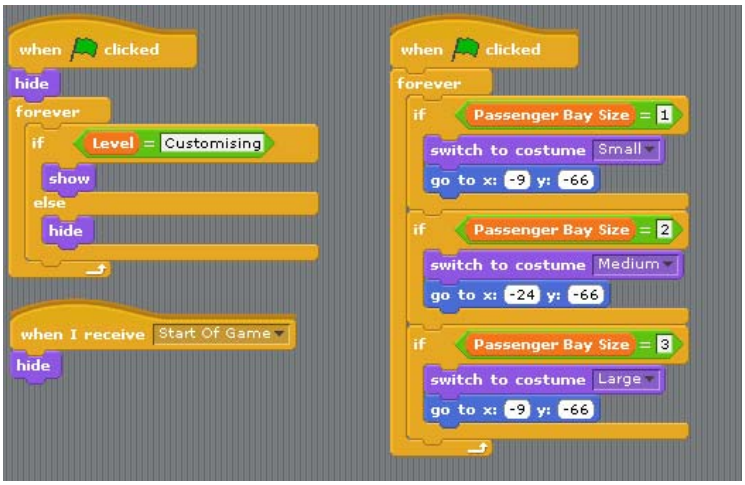
Candidate Number: 0000

Instead of buttons being clicked to customise the features of the robot, I created a variable with a slider, allowing the user to slide between values 1, 2, and 3. I then created a sprite that would change between the words small, medium and large beneath, to show to the user what affect their choice had on the robot. I did this not only for display, but for efficiency, as making additional sprites would be unnecessary and time-consuming, compared to using the variable itself as the interactive component in the Customising Menu.

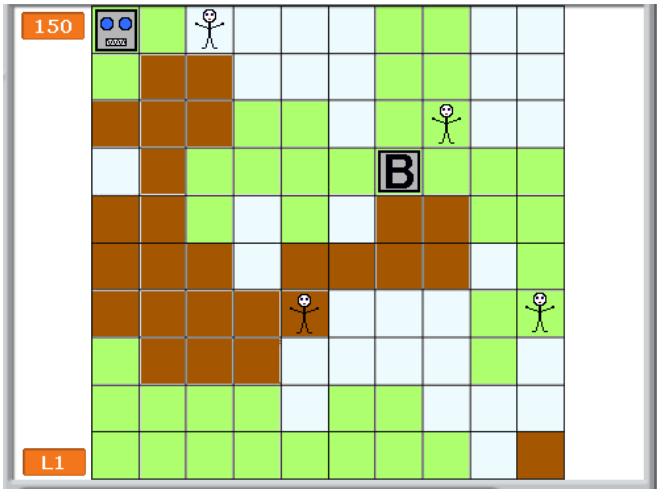
I then created a second variable, for the Traction Type, and a sprite that would change its costume accordingly. The overall customisation menu works like the following (although the 'power units' variable still needs to be hidden):



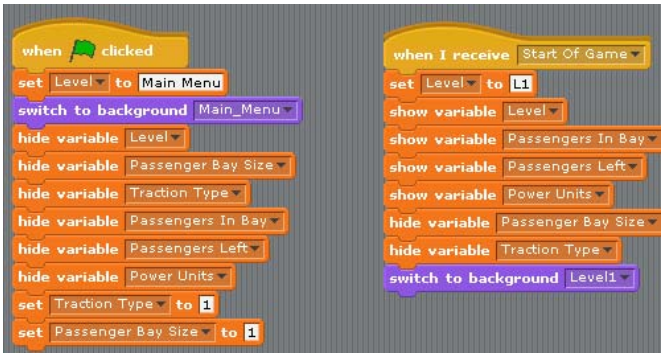
Here is an example of the code used – this is for the Passenger Bay Size sprite, although the Traction Type sprite works in a similar fashion.



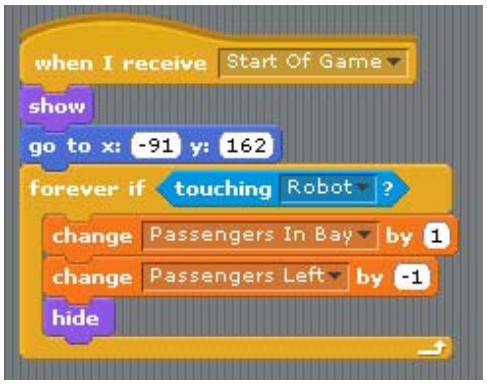
Having finished the menu (for now), I began work on the game itself. I had already made the robot, person and base sprites, as well as where they start on the grid. I had also finished the robot's arrow key movement, so my next task was to add the rest of the people that need to be picked up.



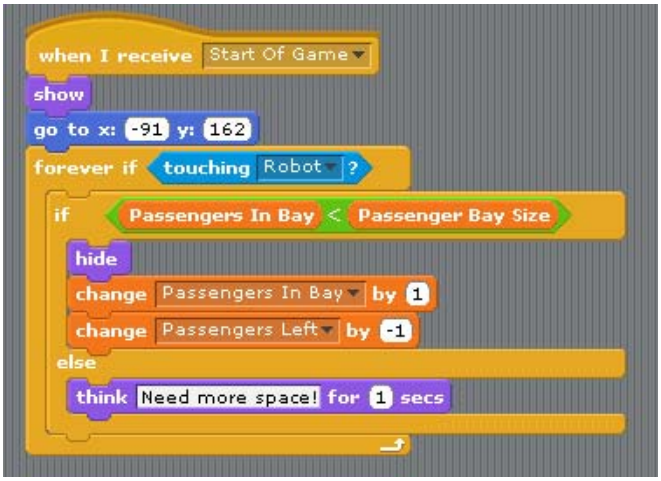
After adding variables for the passengers in the bay and the passengers left on the grid, I then needed to update the code that happens at the very beginning of the game, as otherwise the main menu would display these variables at the start, when they really need to be hidden. This code appears in the Stage script:



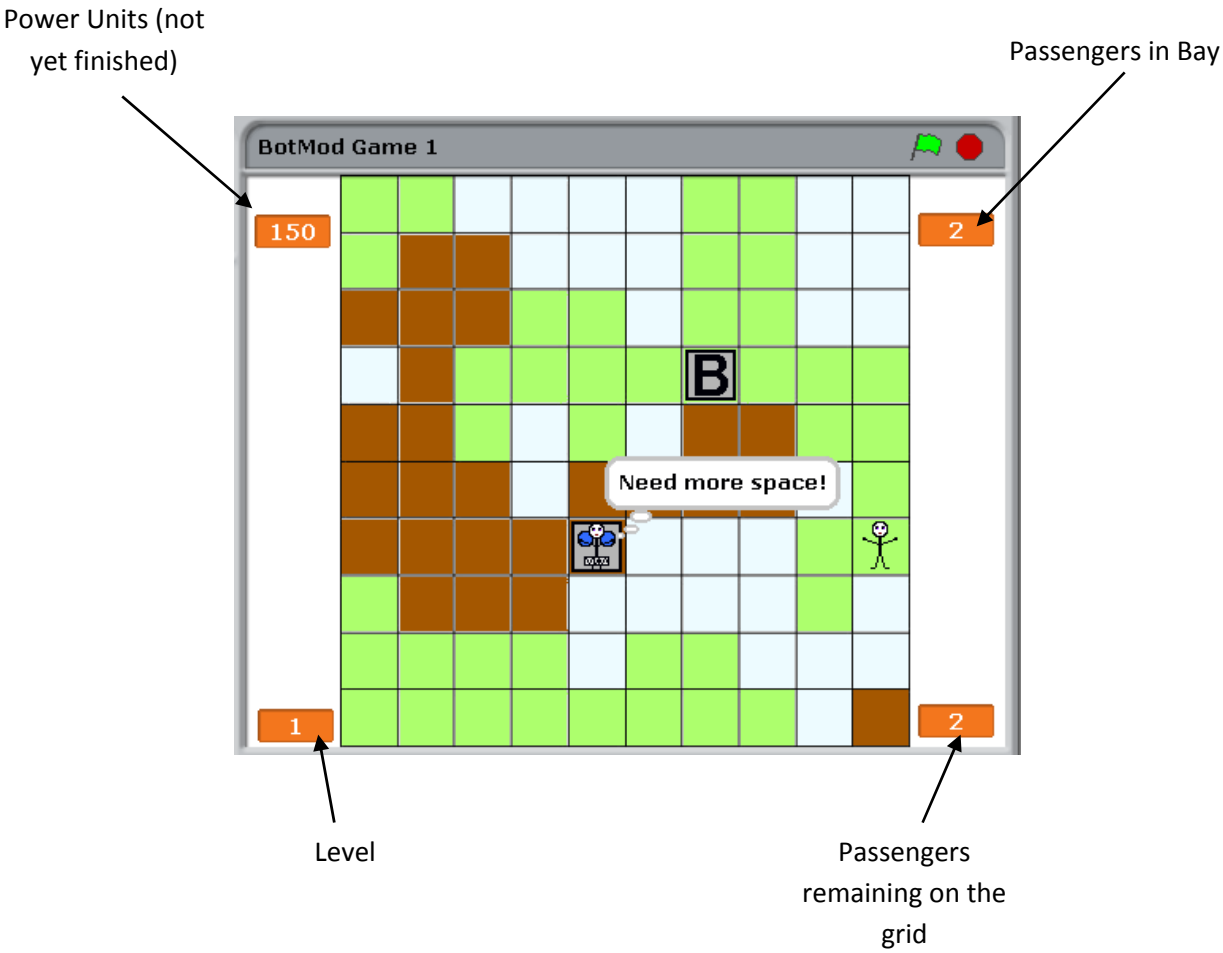
I then began the code for picking up the passengers. I only did this code for Person_1 to begin with, in the knowledge that it would need fine tuning later before it can be duplicated to the other sprites.



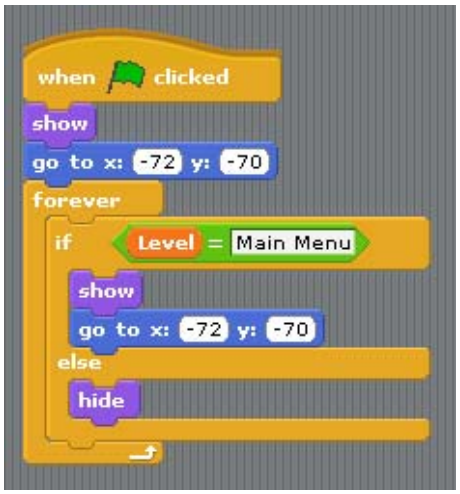
I realised soon after that not only did the person not hide straight away, but that even when the maximum amount of passengers were already present in the passenger bay, the robot would still pick up the person. I used an 'If' statement, asking whether the passengers in the bay were less than the size of the bay. If this was true, then the person could be picked up. If not, a message would be displayed:

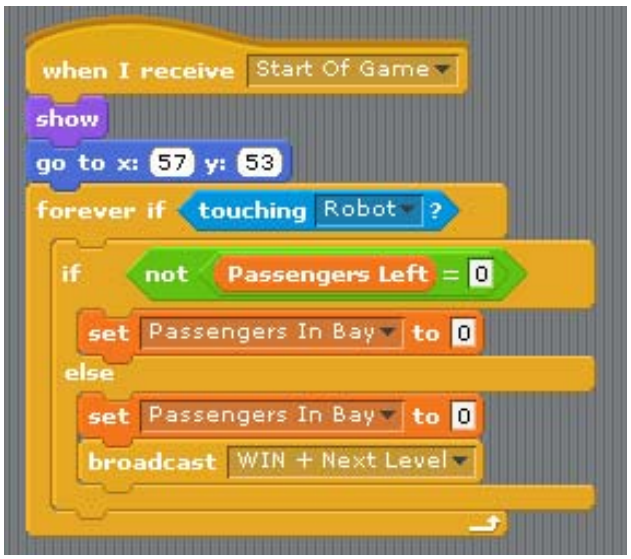


I finished the passenger code, and the result on screen is the following:



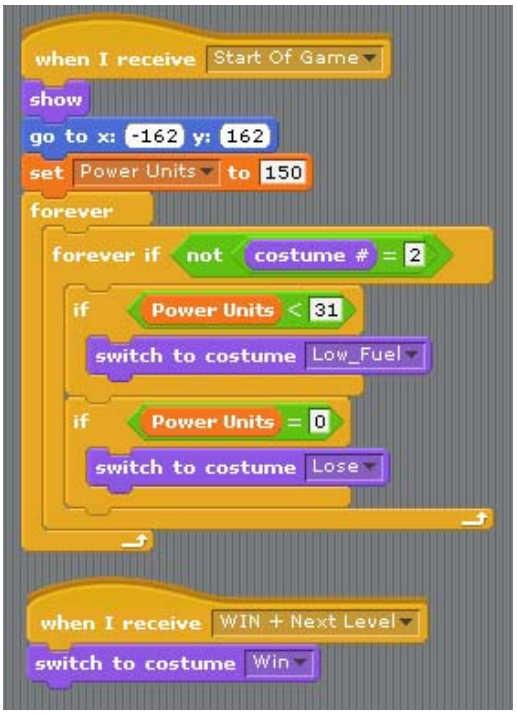
After adding a show command in the Customise Robot button, to prevent it from hiding every other time the game is played, I then began the code for when the robot touches the base, to unload its passengers. I did a 'forever if' piece of code, stating that when the robot touches the base, it will check whether the passengers left on the grid equals 0. If this isn't the case, then the robot's passengers will be unloaded to the base and the passengers-in-bay variable will be reset to 0. However, if there are no more people left to be picked up, then the player has completed the level, which triggers a broadcast for all sprites to receive, indicating the next level.





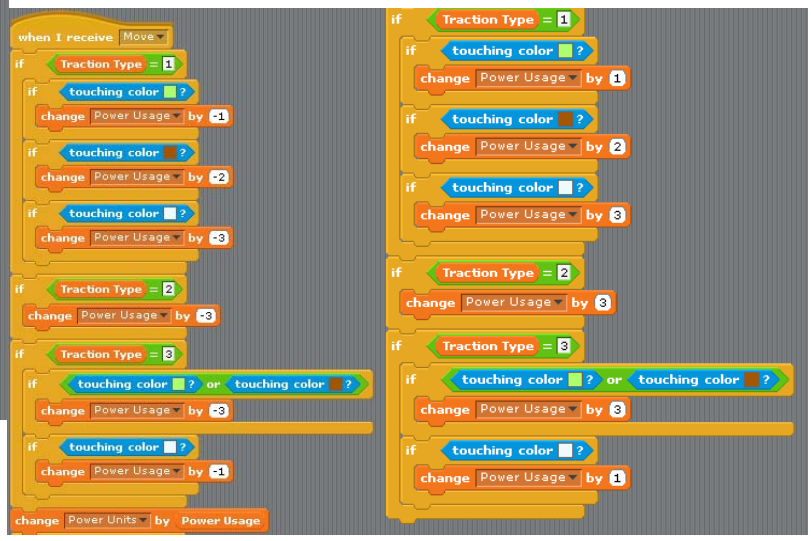
My next objective was to complete the 'power units' variable, as at the time the variable merely set to 150 as instructed at the beginning of the game. However, I first concentrated on changing the robot sprite's costume, to have appropriately coloured eyes whenever a level is completed, or the robot has little or no fuel.

At first, I did a 'forever if' statement, checking at any moment if any of the arrow keys have been pressed, however, although the code was as efficient as possible, and suited the table, the variable increased and decreased by ridiculous amounts.



This was the original code, but I soon realised that the forever if caused these power usage variable changes to occur several times, just as an arrow key is being pressed! My solution was the following: I added a broadcast command into each piece of arrow key movement code, which when received, would only perform the operations on the variables once. However, this power usage variable constantly became a greater and greater value, which each time would be subtracted from the 'power units' variable *here*. This meant that I had to duplicate the new code to add on whatever value the 'power usage' variable was every time. However, even then, the code was far too long, and could easily be more efficient.

The code on the left determines the amount of the 'power usage' variable, which is then taken away from 'power units'. The operation is then performed, and the same value is then added on to power usage, to prevent it from increasing everytime the robot moves. This code was the most robust option to suit the plans for both Lava and Walls, as otherwise the code could be easily broken. Using 2 variables to deal with the power was necessary and easier to control and manage when editing the game later on.

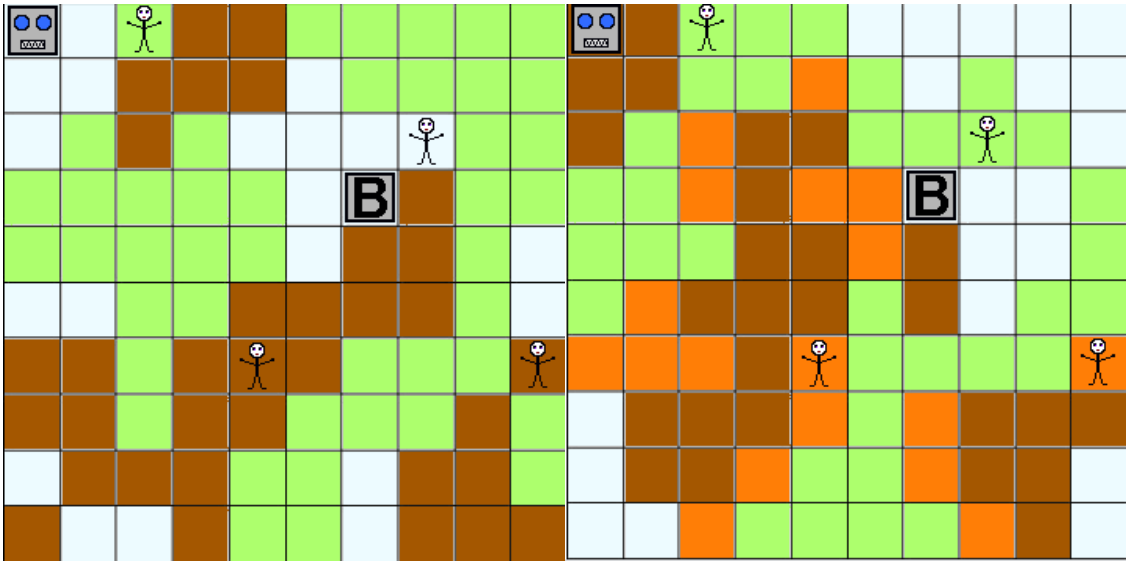


<u>Name: Student 1 Gaming</u>

<u>Candidate Number: 0000</u>

After finishing the power units, I discovered a small problem that needed to be fixed. The robot's eyes are supposed to go red when the power units run out, however the code for this meant that when the power units go lower than 0, the robot never changes sprite.

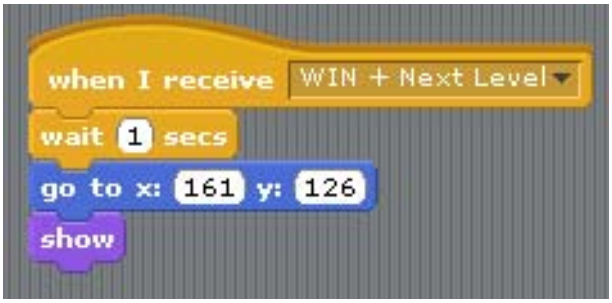
Considering that nearly all of the game is now finished, other than the losing and winning aspects of the game, I thought it was time to add extra levels. Although the people don't yet move to other positions, I created a second and third level:



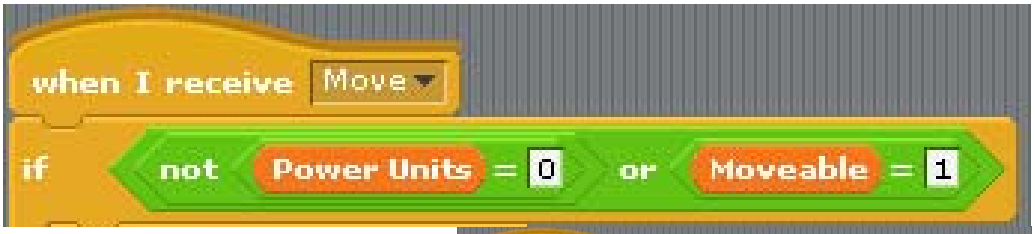
I then concentrated on the progression to the next level. As shown before, a broadcast "WIN + Next Level" will occur, which every sprite receives, performing specific bits of code as a result.

ROBOT:

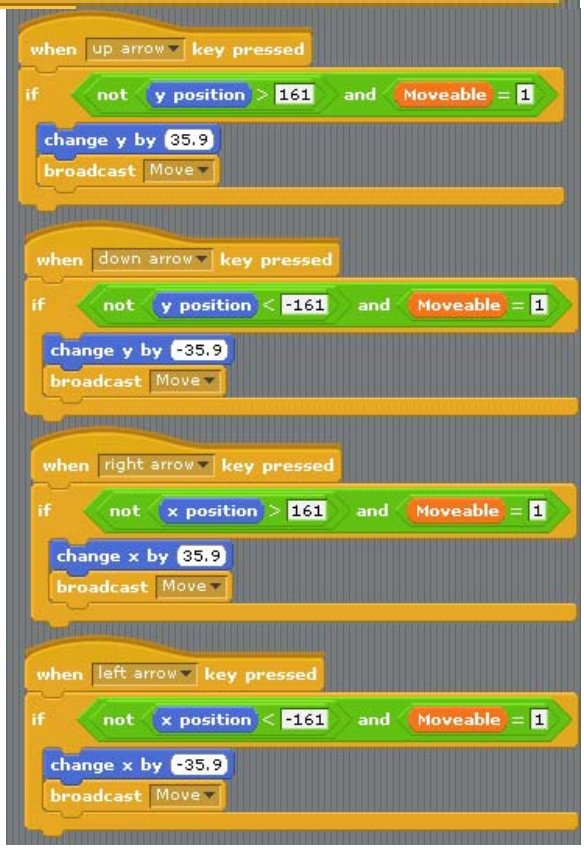
PERSON:

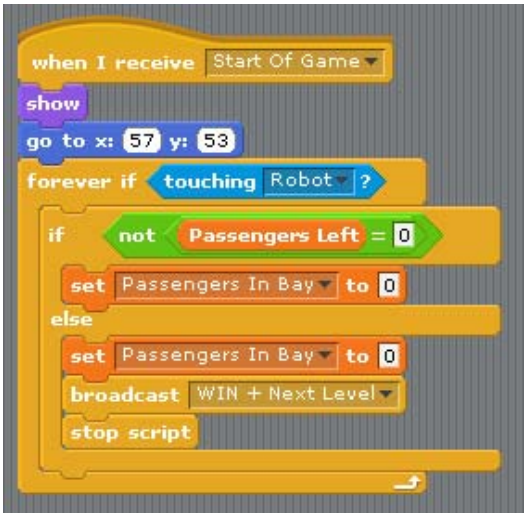


This led me to a problem. When the level was completed, I added a deliberate 1 second delay, before the next level appears. However, this meant that the user could still move the robot and reduce its power units to 0 in between. To rectify this, I created a 'moveable' variable, which would also prevent the player from moving the robot during the main menu. This are 2 examples of the variable, used in the 'power units' and arrow key movement code.



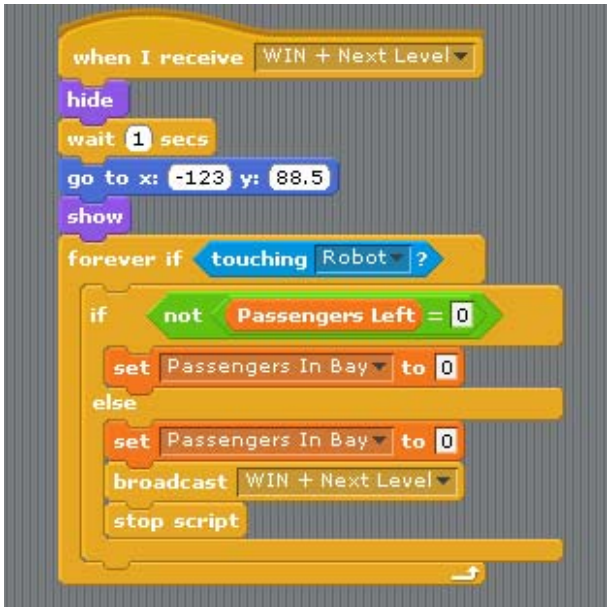
In the next level, I decided that the base and people would move to different positions. As shown on the previous page, the person sprite moves to a new position for level 2, but the Base wouldn't interact with the robot on this level. This was due to a 'stop script' command I used at the end of the forever if, which caused the condition of touching the robot to stop being checked. I did this to prevent the broadcast command repeating itself, as this stopped the next level from happening completely.



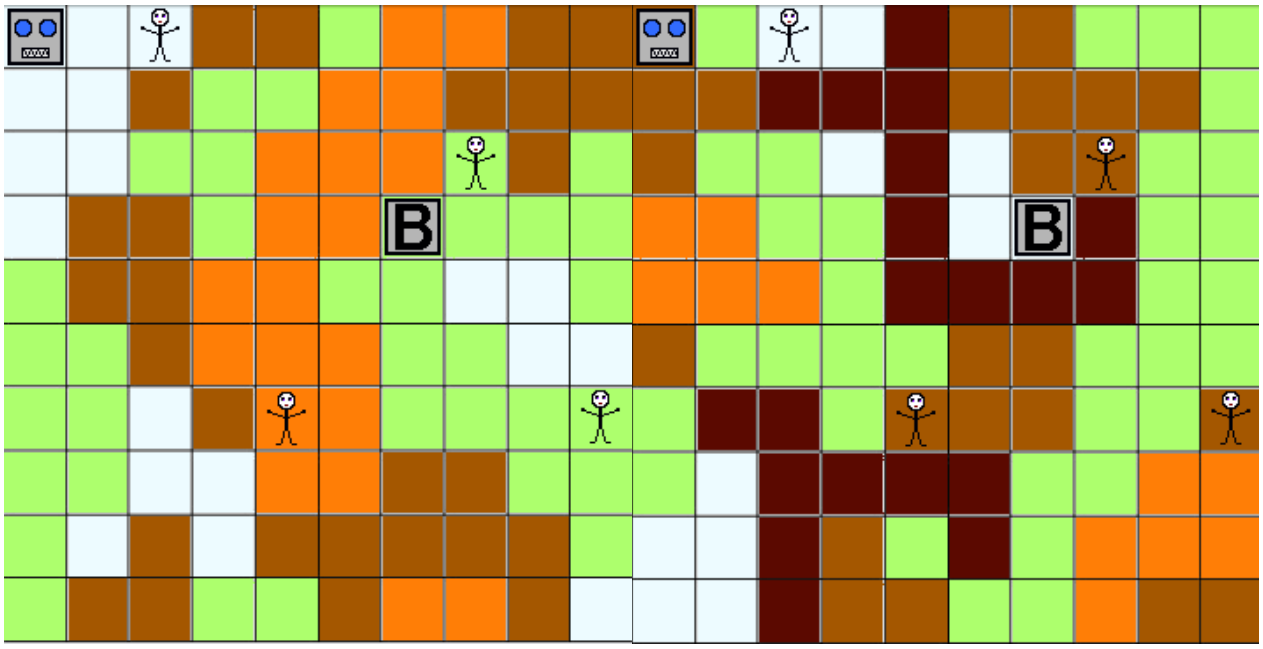


This worked until the player got to the 2nd level, so I had to make a piece of code for its own broadcast. This stated that when the broadcast I received, the base would wait the same time as everything else, 1 second, before going to a new position , therefore no longer touching the robot. This also meant that the robot could move and the forever if statement was then repeated.

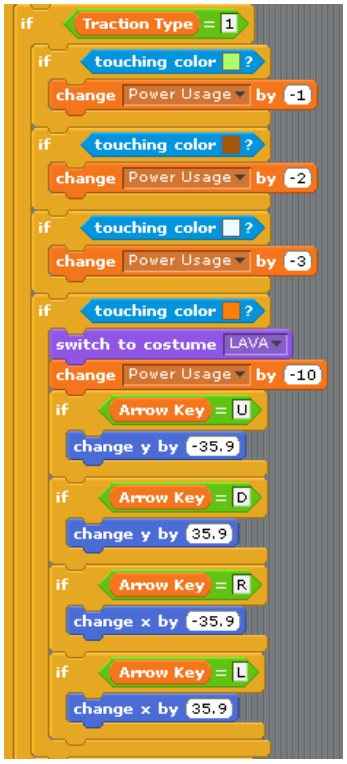
This is the code mentioned on the previous page, although I again had to change the code based on the fact that I had added other levels. This was because every level the base would return to the same position, which ended up in the level variable constantly increasing every second. So in the finished version, there were 4 'go to' statements, 1 for each time the next level is achieved. Of course, when the game is won, the base needs to be hidden.



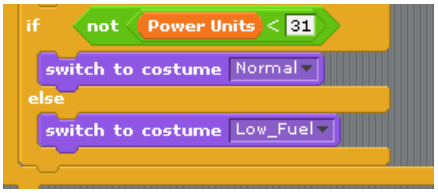
I then added the 4th and 5th level, which looked like the following:

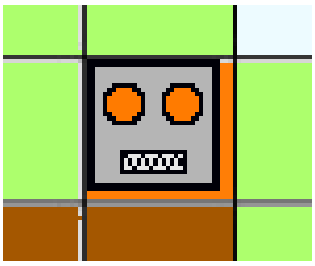


The 4th level I designed to add a new strategic element, where unless the tracks are in use, the player is not able to cross the landscape without crossing the rocks at the bottom. I did this as the tracks are otherwise a disadvantage to the player, therefore they would come in handy for this level in particular and any other option would make the level extremely difficult. The 5th level however, features walls, which hopefully would be unable to pass through, when I would code it later in the game.



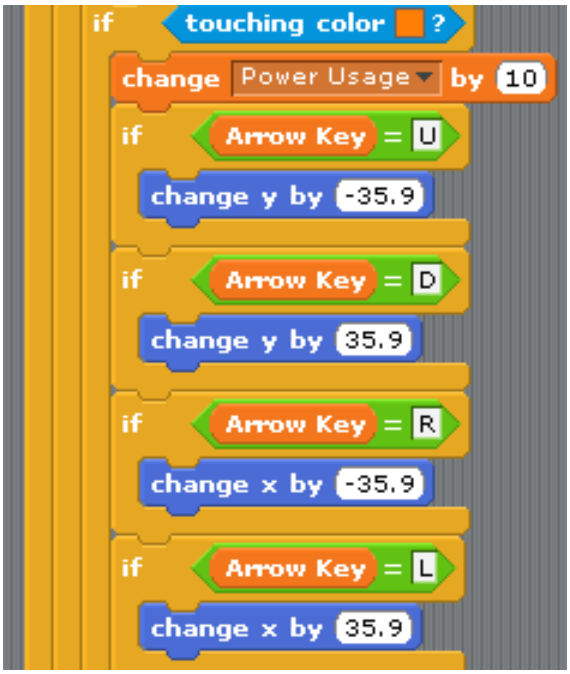
This is an extract from the updated version of the power usage system. By adding lava, the result was that the robot did not react with the orange colour in any way; therefore I had to add a new piece of code for the contact with lava. This is used when the wheels are equipped (when Traction Type = 1). The power usage will decrease by a full 10 units, and I created a new variable for what arrow key is being pressed when the move is made. This was done so that, for example, if the robot enters the lava from the left, the robot will be thrown back in that direction, to prevent it from passing the lava. I later added a wait command, for 0.5 seconds, just before the arrow key 'if' statements, so that the costume was visible, as otherwise the costume would immediately change back to normal, thanks to the lower half of this piece of code.



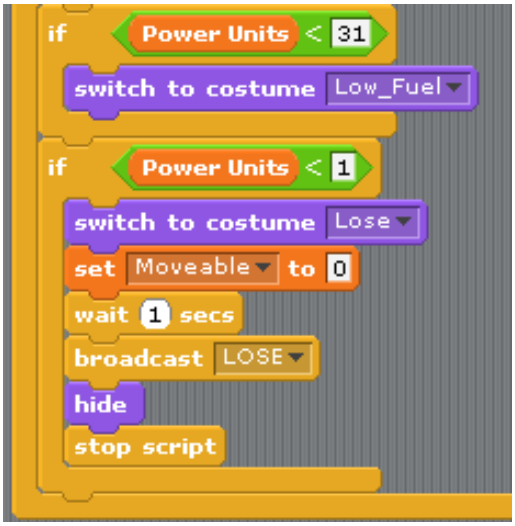


This is the result. Although the robot is slightly off the grid (which I couldn't quite understand) the robot waited or half a second in this position, before returning to the square above, with 10 less power units and its normal costume. However, a glitch occurred with a later part of the code, and the power units wouldn't change and even started increasing! Therefore the arrow key 'if' statements were moved to after the "change 'Power Units' by 'Power Usage'" command. Therefore the -10 on power usage would be countered after the above operation happens. This worked a lot easier, and even before this change, the tracks were able to pass through the lava as planned.

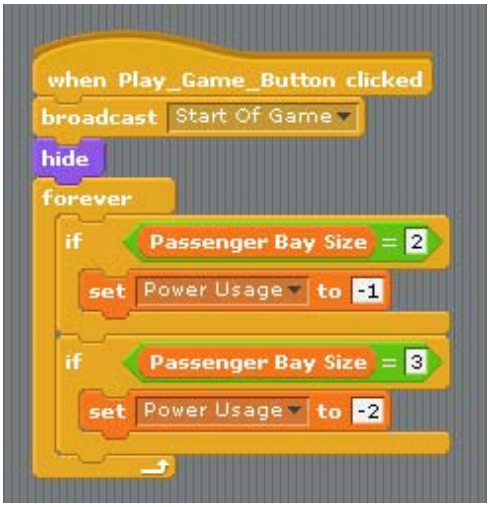
After this problem, I focused on the final area of my game – the winning and losing sequences. I had the screens already sorted, but some form of broadcast needed to happen, for all sprites to hide etc.



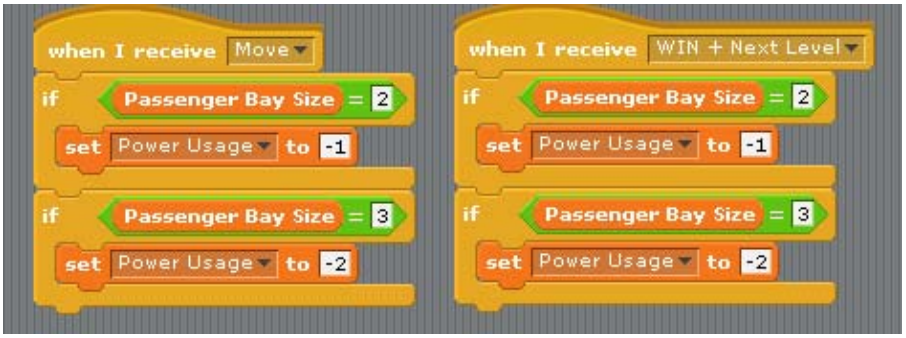
The LOSE broadcast works as follows: When the power units reach 0, the costume changes to the red-eyed robot, and after a 1 second wait, the LOSE broadcast occurs. I had to find an alternative to part of the Play Game script in order to make the sprite show every time the main menu is present, as otherwise at the start of the game the sprite wouldn't hide when the Play Game object is clicked.



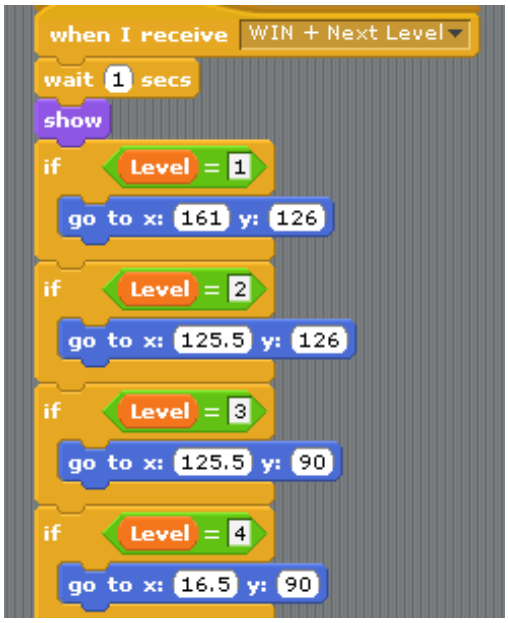
Another problem I encountered was the passenger bay size, which turned out to have no effect on the power units whatsoever! Originally the code was that of the screenshot on the left, however once the power usage variable was set, another part of code somewhere would instantly set it back to 0 again in preparation for the start of the first level. This meant that no extra power units were lost for choosing the medium or large passenger bays. The code on the right was the obvious solution.



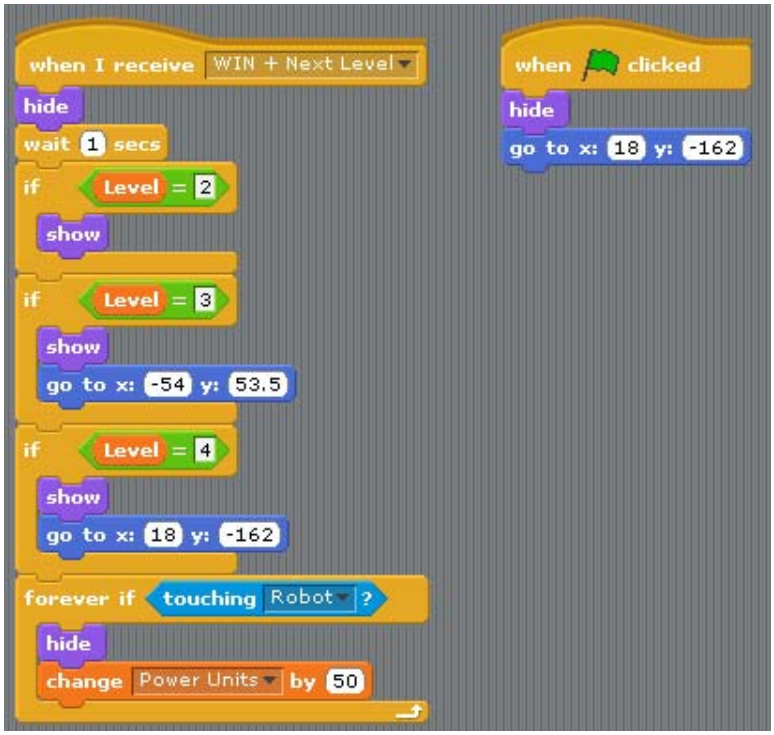
However, this resulted in the lava not working, as there was a deliberate ½ second delay for effect, which meant that the power usage would be set back the size of the passenger bay before the lava's code had the chance to take place. Therefore I had to add the following:



The game had everything it needed, apart from the code for the walls on the final level. However, I realised that the levels I had constructed were practically impossible to complete using the same configuration. This was partially due to the fact that I had forgot to change the positions of the passenger sprites, so on each level they stayed at the same point on the grid, resulting in one of them swimming in lava on level 4! I created this code to solve the problem (the 'go to' commands were different for each individual).

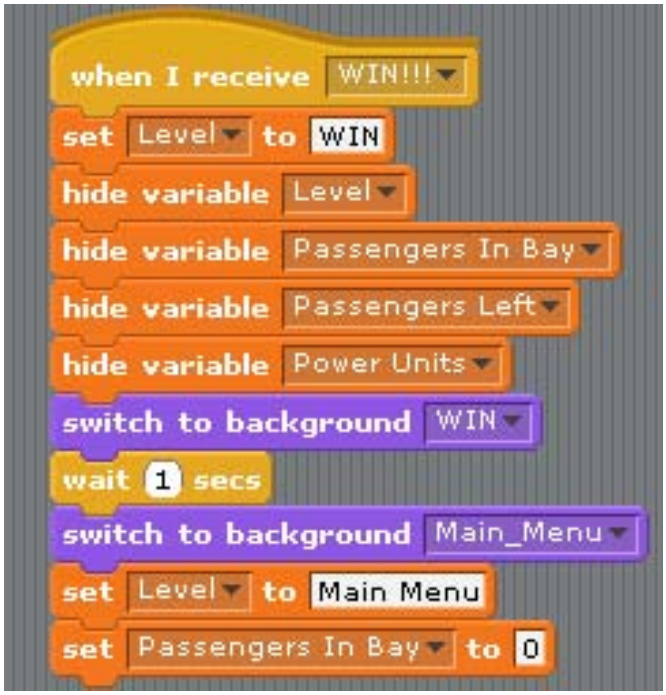


After playing the game, I discovered that particularly on level 3, the game became insanely difficult and, in many configurations of the robot, impossible. Therefore I had to add a Fuel Can that would add an additional 50 points to the power units. This worked like the following:



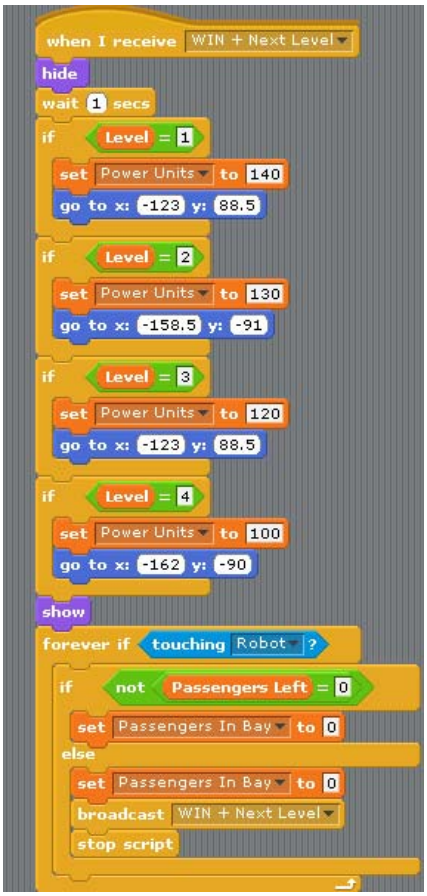
The value by which the power units get changed, however, still ended up not great enough for the final levels, so I used the following expression:

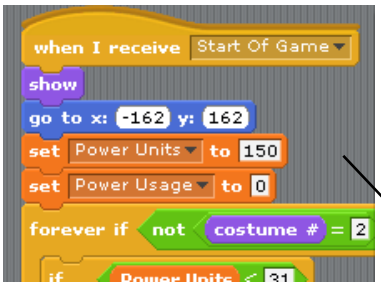
My final step was to code the walls. These were built for the last level and beating this would complete the game. The game, when won, worked like this:



Obviously, the other sprites would also receive this broadcast, alerting them to hide until the game is replayed. All the 4 on-screen variables get hidden, and the code generally works in a similar fashion to the Lose broadcast.

The task states that the power units must start at lower values as each level occurred. Therefore with my Fuel Can finished, I was able to do this without making any level impossible. This was the last step I took, before I realised that the game was now finished, and I was able to complete the game, or repeat it if I lost. All bugs that had been a part of the game I quickly fixed (as shown below) and I moved on to the Techniques Used and eventually the Testing of the game.





Techniques Used.

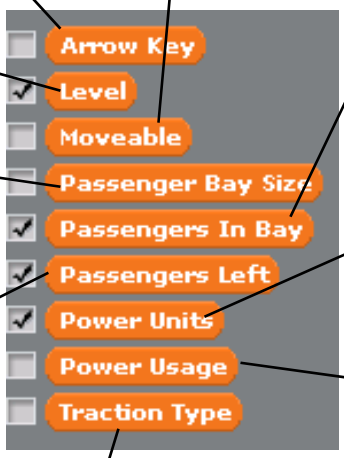
This is the very first piece of code the robot will follow. I used the 'When Flag Clicked' control command in order for the following code to take effect as soon as the game is started. The robot will then switch to its beginning costume that is uses for the majority of the game, and then hides, as before the game is started, the user must go through the Main Menu screen.

This variable is set at any point to U, D, L or R, to represent Up, Down, Left or Right. This was added toward the end of the game to know where the robot needs to move after making contact with Lava or a Wall from Level 3 and above.

This Moveable variable was added to prevent the robot from being moved at any time by setting the variable to 0. This is needed, for example, before the game started as otherwise the player could pick up passengers and use up power units before the game begun.

The Level variable is both for display purposes, as well as to keep track of which level was being played, or if the user was on the Main Menu, in which circumstance this variable is hidden.

This variable keeps track of the passengers currently in the passenger bay at any given moment during the game.



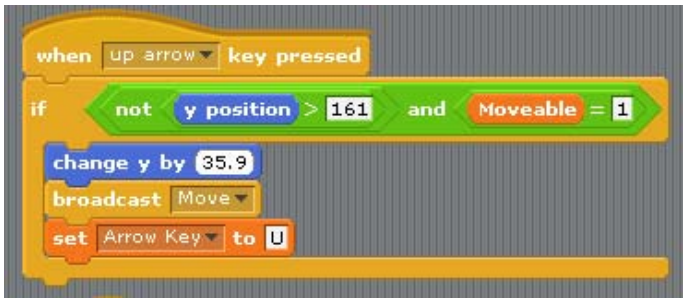
The Passenger Bay Size is simply the variable that keeps track of the choice of passenger bay the player has made. This had an impact on the power units used by the robot, and the amount of passengers that could be picked up. This was displayed on the Customising Menu.

This is a very important variable, the power units. This is set to 150 at the first level, and is subtracted by the Power Usage variable every time the Robot moves. This is used on screen during the game and when this reaches 0, the player loses the game.

The Passengers Left variable is on display throughout the game, and informs the user how many passengers are left on the grid.

This is also very related to the Power Units variable, as this is taken away from the Power Units every time the player moves the robot. This is the accumulated total of the effects of the Traction Type, Passenger Bay Size and the Terrain that the Robot has moved to.

The Traction Type variable keeps track of the user's choice of Wheels, Tracks or Skis, and is set to 1, 2 or 3 accordingly. This variable is one of the few that have an impact on the amount subtracted from the Power Units.



Many of these variables can be seen throughout the code, so I added them to help the code to be understood more easily. An example of the Moveable variable, for instance, is seen here. This code occurs whenever the Up arrow key has been pressed. As long as the position of the robot allows space above it on the grid, and as long as the Moveable variable is set to 1 (meaning the robot can move), then its Y position will change, the Move broadcast occurs, and the Arrow Key variable is set to U.

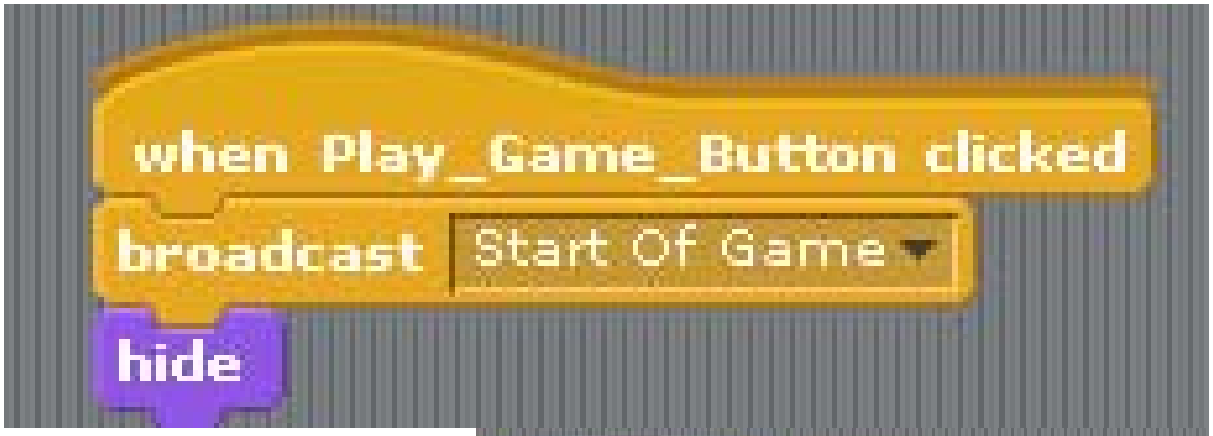
Name: Student 1 Gaming

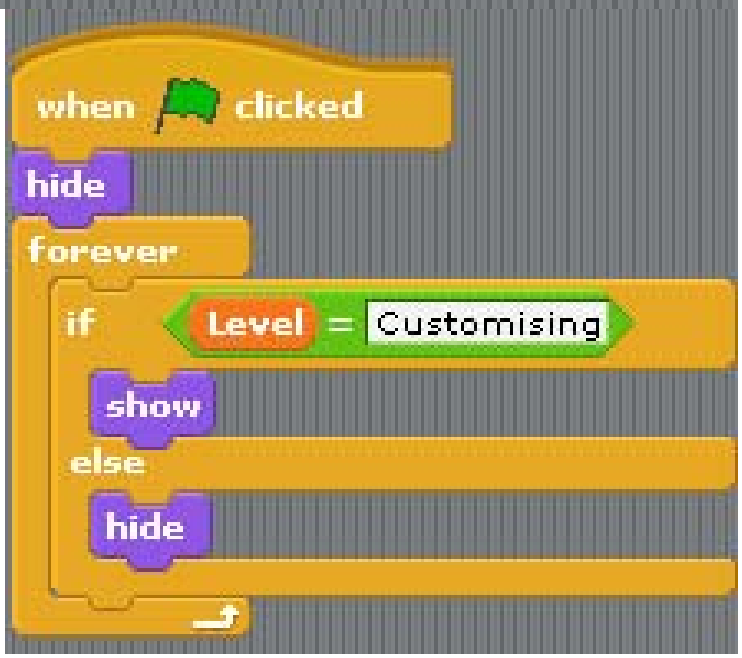
Candidate Number: 0000

This is the Move broadcast in question. As long as the Power Units are more than 0, and the same variable 'Moveable' is set to 1, then the following code can take place. If the Traction Type is set to 1 (meaning Wheels) then, depending on the Terrain, the Power Usage variable will be changed accordingly. This happened for the Tracks and Skis, therefore one of the 3 parts of the code must take place. After this the Power Usage variable (the accumulated total of all the Robot's customisation variables and the current Terrain) is then subtracted from the Power Units.

In my code many broadcasts were used to transmit messages between sprites, as this seemed to be more efficient when used once or when alternative variables already in use within the broadcast. This is a good example, as this broadcast comes into play every time the user completes a level, excluding Level 5. Depending on the current level is in use, the Base will wait, before moving to a new position, to add difficult and a lack of repetition to the structure of the game. Once reappeared, the Base then permanently checks whether the Passengers Left is 0, in which case, when the robot is touching the sprite, the Broadcast will occur again, before stopping, as otherwise the code would repeat and the variables would undergo dramatic changes every time the broadcast occurs, which would happen over and over again.

The next control command I used was for when the Play Game Button is clicked. Another Broadcast, the 'Start Of Game' occurs, before the button hides, only to show again when the game is restarted or lost. This Start Of Game Broadcast is essential to the beginning of the game.





On many occasions the 'Forever' code is used along with 'If' statements, as opposed to just using a simple 'Forever If'. This is done not only to run multiple 'If' statements that will be constantly checked forever, but to make the code more efficient (as the alternative would be to make several 'Forever Ifs' which would take a long time to create and would be difficult for the programmer to follow). No additional code can be added to one 'Forever If' underneath, whereas multiple pieces of code can be stored within a 'Forever'. This simple piece of code is part of the Traction Type sprite, and dictates the moment in which the sprite is shown. Having several 'Forever If' statements would also make the code much less robust, as the pieces of code would be separate instead of being in the same statement.

Game Testing.

1) **Green Flag**
2) **Menu & Customising Screens (+ Progression To Level 1)**
3) **Robot Movement & Passenger/Base Interaction**
4) **Level 2 Background & Sprite Positions**
5) **Level 3 Background & Sprite Positions**
6) **Level 4 Background & Sprite Positions**
7) **Level 5 Background & Sprite Positions**
8) **Win/Lose Screens**

Test Number	Text Description	Input Data/User Action	Expected Result	Actual Outcome	Change Required?
1.1	Main Menu display at start of game.	Green Flag clicked.	Main Menu displayed, along with 'Play Game' & 'Customise' sprites.	Main Menu displayed, along with 'Play Game' & 'Customise' sprites.	**No**
1.2	Variables set accordingly as well as being hidden at start of game.	Green Flag clicked.	Any visible variables hidden, 'Traction Type' & 'Passenger Bay Size' set to 1, 'Passengers In Bay' & 'Power Usage' set to 0.	All variables hidden, 'Traction Type' & 'Passenger Bay Size' set to 1, 'Passengers In Bay' & 'Power Usage' set to 0.	**No**
1.3	Sprites being visible at start of game.	Green Flag clicked.	All sprites within levels should be hidden when game is started. Only 'Play Game' and 'Customise' sprites should be visible.	All sprites except 'Play Game' and 'Customise' are hidden when game is started.	**No**
2.1	Level begins when 'Play Game' is clicked.	'Play Game' sprite clicked.	Background should become Level 1; all appropriate sprites should be shown & hidden – robot sprite should be set to Normal.	Background becomes Level 1; Robot, People and Base sprites shown, whilst Play Game and Customise sprites are hidden. Robot sprite is on Normal.	**No**
2.2	Appropriate variables shown & changed when level begins.	'Play Game' sprite clicked.	Power Units should be set to 150, Power Usage to 0, Level to 1, Passengers Left to 4 and Moveable to 1.	Power Units set to 150, Power Usage to 0, Level to 1, Passengers Left to 4 and Moveable to 1.	**No**
2.3	Customising screen displayed when 'Customise'	'Customise' sprite clicked.	'Level' variable should be set to customise (changing the background); Traction	Background and appropriate variable is changed, while the correct	**No**

	is clicked.		Type, Passenger Bay Size and Play Game sprites should be displayed along with the Customise sprite being hidden.	sprites that assist the user's customisation process are shown. Customise sprite hidden, Play Game sprite moved to bottom and still showing.	
2.4	Traction Type variable slider.	Slider display for Traction Type variable being changed by user.	When T.T variable set to 1, T.T sprite should become wheels. When T.T variable set to 2, T.T sprite should become tracks. When T.T set to 3 by user, T.T sprite should depict skis.	At 1, the T.T sprite depicts wheels, at 2, tracks are displayed and when the T.T variable is changed to 3, the sprite below becomes skis.	**No**
2.5	Passenger Bay variable slider.	Slider display for Passenger Bay Size variable being changed by user.	When P.B.S variable set to 1, P.B.S sprite should show the word 'Small'. When P.B.S variable set to 2, P.B.S sprite should say 'Medium'. When P.B.S set to 3 by user, P.B.S sprite should display 'Large'.	When P.B.S variable set to 1, P.B.S sprite displays the word 'Small'. When P.B.S variable set to 2, P.B.S sprite displays 'Medium', and when P.B.S set to 3 by user, P.B.S sprite displays 'Large'.	**No**
2.6	Level begins when 'Play Game' is clicked on Customising screen.	'Play Game' sprite clicked (in Customising screen).	Background should become Level 1; all appropriate sprites should be shown & hidden – robot sprite should be set to Normal.	Background becomes Level 1; Robot, People and Base sprites shown, whilst Play Game, Traction Type and Passenger Bay Size sprites are hidden. Robot sprite is on Normal.	**No**
2.7	Appropriate variables shown & changed when level begins.	'Play Game' sprite clicked (in Customising screen).	Power Units should be set to 150, Power Usage to 0 (unless Passenger Bay Size causes a necessary change), Level to 1, Passengers Left to 4 and Moveable to 1. T.T & P.B.S slider variables should disappear.	Power Units set to 150, Power Usage to 0 (if P.B.S changed to 2, P.Usage becomes -1, if P.B.S set to 3, P.Usage becomes -2), Level to 1, Passengers Left to 4 and Moveable to 1. T.T & P.B.S slider variables disappear.	**No**
3.1	Robot up/down/left/right movement.	Up, Down, Left or Right arrow key pressed.	'Arrow Key' variable set to U, D, L or R accordingly; for Up, y-axis of Robot should change by 35.9 (1 square), for Down, -35.9, for Left, x-axis change by -35.9, for Right, 35.9.	'Arrow Key' variable set to U, D, L or R accordingly; for Up, y-axis of Robot changes by 35.9, for Down, -35.9, for Left, x-axis changes by -35.9, for Right, 35.9.	**No**

3.2	Robot on edge of grid.	Up, Down, Left or Right arrow key pressed (while on edge of grid).	'Arrow Key' still set to direction of arrow key pressed, however Robot should not move if the direction of travel would send it off the grid.	'Arrow Key' still set to direction of arrow key pressed, however Robot does not move when the movement would have led the sprite out of grid.	**No**
3.3	Power Unit change when moving.	Up, Down, Left or Right arrow key pressed.	The following must be added to the possibly negative value of the P.Usage variable (See **3.0**): <u>For wheels:</u> If on grass, P.Usage = -1, If on rock, P.Usage = -2, if on ice, P.Usage = -3. <u>For tracks:</u> P.Usage = -3. <u>For skis:</u> If on grass or rock, P.Usage = -3, if on ice, P.Usage = -1.	The expected outcome is met, then as expected the total of the Power Usage variable is subtracted from the Power Units (all taking place within a Move Broadcast piece of code).	**No**
3.4	Sprite costume changes.	When Power Units are low/gone or when winning a level.	At start of game and under normal conditions, the robot should have blue eyes. When a level is completed, the eyes should be green. When the robot makes contact with lava, orange eyes should occur, when the power units are low, the eyes should be purple and when power units are 0, red eyes are the result.	At start of game and under normal conditions, the robot has blue eyes. When a level is completed, the eyes temporarily go green. When the robot makes contact with lava, orange eyes occur, when the power units are low, the eyes are purple and when power units are 0, red eyes are the result.	**No**
3.5	Passenger interaction (with space).	Robot being moved to the same square as a passenger when there is space in the passenger bay.	If space is available within the Passenger Bay, the person can be picked up – the sprite will hide, and variables should change accordingly.	The person disappears, adding 1 to the Passengers In Bay variable, and taking 1 away from the Passengers Left variable.	**No**
3.6	Passenger interaction (without space).	Robot being moved to a passenger when	With no extra space in the P.B, the person will stay put – the robot will display	Only the Power Unit variable changes, and the intended message is	**No**

		there is no space in the passenger bay.	a 'Need More Space!' message and no variables regarding the passenger count will change.	displayed until the user moves the Robot out of the way.	
3.7	Base interaction (without finishing the game).	Robot making contact with the Base.	If there are passengers present, then they will be taken from the passenger bay (until Passengers Left = 0). Nothing will occur when contact is made with the Base without a person.	The passengers are loaded into the Base, being taken away from the P.I.B variable, and without passengers, nothing other than the P.U variable changes.	**No**
3.8	Contact with Lava or Walls.	Robot making contact with either the Lava or the Walls (Walls on Level 5 only).	When the user runs into Lava, the eyes become orange and, following a short delay, the robot will return to its former position, using the Arrow Key variable to determine the original position. When the Robot makes contact with a wall, it instantly gets pushed back using the same variable to the same position.	When the user runs into Lava, the eyes become orange and, following a short delay, the robot returns to its former position, using the Arrow Key variable as expected. When the Robot makes contact with a wall, it instantly gets pushed back (using the same variable) to the same position.	**No**
3.9	Contact with Fuel Can sprite.	Robot making contact with the Fuel Can.	On level 3, the Fuel Can should display for the first time and offer the Robot 50 power units, 100 on Level 4 and 150 on Level 5. These points are allocated when contact with the sprite is made, causing the sprite to disappear.	On level 3, the Fuel Can displays for the first time and offers the Robot 50 power units, 100 on Level 4 and 150 on Level 5. These points are allocated when contact with the sprite is made, causing the sprite to disappear.	**No**
4.1	Progression to Level 2.	Robot making contact with Base when P.L = 0.	When the user delivers the final passenger/s to the Base (with power units of course), its eyes should turn green for a short amount of time when everything freezes, before the next level appears onscreen.	When the user delivers the final passenger/s to the Base with power units, its eyes turn green for a short amount of time when everything freezes, before the next level appears onscreen.	**No**
4.2	Passengers reappear in new	Progressing to next level.	After the short pause to represent the level's completion, the 4	After the short delay, all 4 passengers reappear in new positions that Level 2	**No**

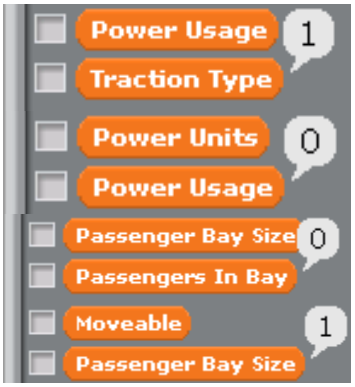
	positions.		passengers should reappear in new positions that are expected for Level 2.	is supposed to be accompanied by.	
4.3	Base and Robot appear in new positions.	Progressing to next level.	The Robot and Base sprites should appear in new positions.	The Robot and Base sprites appear in new positions.	**No**
4.4	Variables reset – Background changes.	Progressing to next level.	Power Units should be set to 140, Passengers Left to 4 and Level to 2.	Power Units are set to 140, Passengers Left to 4 and Level to 2.	**No**
5.1	Progression to Level 3.	Robot making contact with Base when P.L = 0.	When the user delivers the final passenger/s to the Base (with power units of course), its eyes should turn green for a short amount of time when everything freezes, before the next level appears onscreen.	When the user delivers the final passenger/s to the Base with power units, its eyes turn green for a short amount of time when everything freezes, before the next level appears onscreen.	**No**
5.2	Passengers reappear in new positions.	Progressing to next level.	After the short pause, the 4 passengers should reappear in new positions that are expected for Level 3.	After the short delay, all 4 passengers reappear in new positions that Level 3 is supposed to be accompanied by.	**No**
5.3	Base and Robot appear in new positions. Fuel Can appears for the first time.	Progressing to next level.	The Robot and Base sprites should appear in new positions, and the Fuel Can should make its first appearance toward the bottom of the grid.	The Robot and Base sprites appear in new positions, and the Fuel Can makes its first appearance toward the bottom of the grid.	**No**
5.4	Variables reset – Background changes.	Progressing to next level.	Power Units should be set to 130, Passengers Left to 4 and Level to 3.	Power Units are set to 130, Passengers Left to 4 and Level to 3.	**No**
6.1	Progression to Level 4.	Robot making contact with Base when P.L = 0.	When the user delivers the final passenger/s to the Base (with power units of course), its eyes should turn green for a short amount of time when everything freezes, before the next level appears onscreen.	When the user delivers the final passenger/s to the Base with power units, its eyes turn green for a short amount of time when everything freezes, before the next level appears onscreen.	**No**
6.2	Passengers reappear in new	Progressing to	After the short pause, the 4 passengers should	After the short delay, all 4 passengers reappear in	**No**

	positions.	next level.	reappear in new positions that are expected for Level 4.	new positions that Level 4 is supposed to be accompanied by.	
6.3	Base, Robot and Fuel Can appear in new positions.	Progressing to next level.	The Robot, Base and Fuel Can sprites should appear in new positions.	The Robot, Base and Fuel Can sprites appear in new positions.	**No**
6.4	Variables reset – Background changes.	Progressing to next level.	Power Units should be set to 120, Passengers Left to 4 and Level to 4.	Power Units are set to 120, Passengers Left to 4 and Level to 4.	**No**
7.1	Progression to Level 5.	Robot making contact with Base when P.L = 0.	When the user delivers the final passenger/s to the Base (with power units of course), its eyes should turn green for a short amount of time when everything freezes, before the final level appears onscreen.	When the user delivers the final passenger/s to the Base with power units, its eyes turn green for a short amount of time when everything freezes, before the final level appears onscreen.	**No**
7.2	Passengers reappear in final positions.	Progressing to next level.	After the short pause, the 4 passengers should reappear in new positions that are expected for Level 5.	After the short delay, all 4 passengers reappear in their last positions that Level 5 is supposed to be accompanied by.	**No**
7.3	Base, Robot and Fuel Can appear in new positions.	Progressing to next level.	The Robot, Base and Fuel Can sprites should appear in their final positions.	The Robot, Base and Fuel Can sprites appear in their final positions.	**No**
7.4	Variables reset – Background changes.	Progressing to next level.	Power Units should be set back to 150 (for final level), Passengers Left to 4 and Level to 5.	Power Units are set to 150, Passengers Left to 4 and Level to 5.	**No**
8.1	Winning The Game!	Touching Base on Level 5 with Passengers Left at 0.	Win screen should be displayed for a short period of time, before returning to Main Menu with all variables and sprites reset to default positions.	Win screen is displayed for a short period of time, before returning to Main Menu. However, although variables were reset, some of the sprites in the game reappeared at the Main Menu, meaning the Next Level code had occurred.	**Added 'If Not Level = 5' for the code that starts each new level. Changed other sprites' code appropriately.**
8.2	Losing The Game ☹	Running Out Of Power Units before	Lose screen should be displayed for a short period of time, before	Lose screen is displayed for a short period of time, before returning to the	**No**

		completing a level.	returning to the Main Menu with all variables and sprites reset to default positions.	Main Menu with all variables and sprites reset to default positions.	
8.3	Replaying The Game	Clicking Play Game & Customise sprites, etc.	All variables and sprites should act as if the game had just been started. All in-game sprites should be hidden, and all variables should return to their original setting.	All variables and sprites act as if the game had just been started. All in-game sprites are hidden, and all variables return to their original setting.	**No**

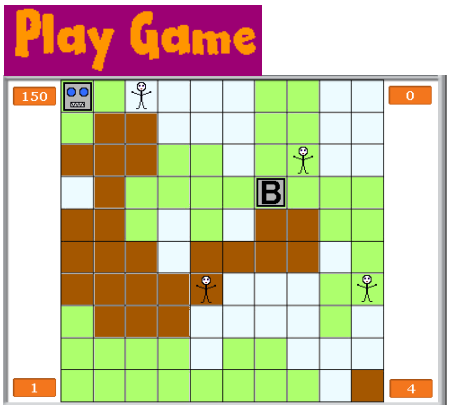
1.1

1.2

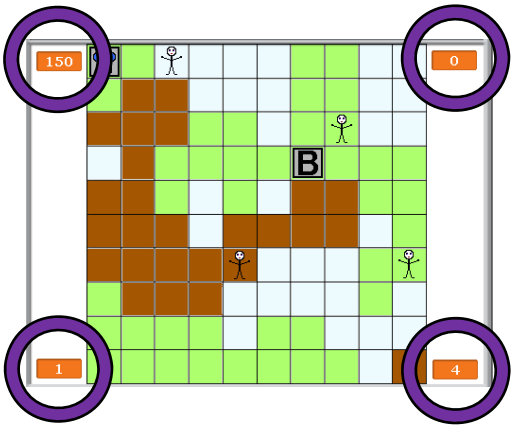


1.3

2.1



2.2



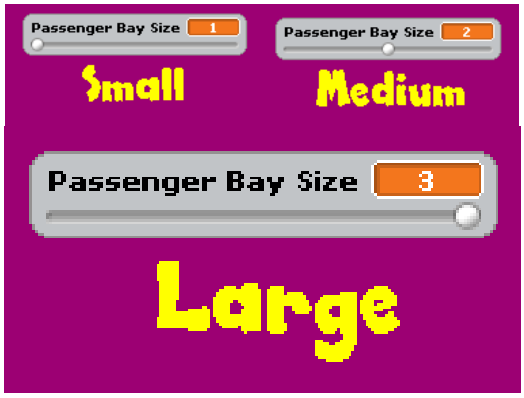
2.3

Name: Student 1 Gaming

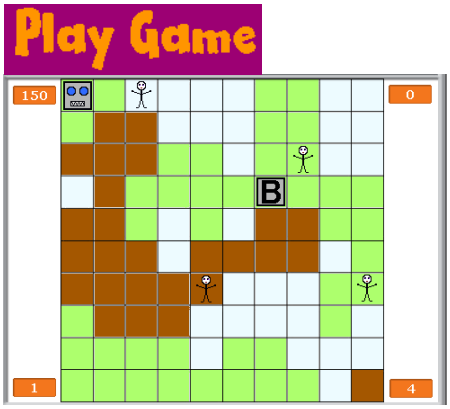
Candidate Number: 0000

2.4

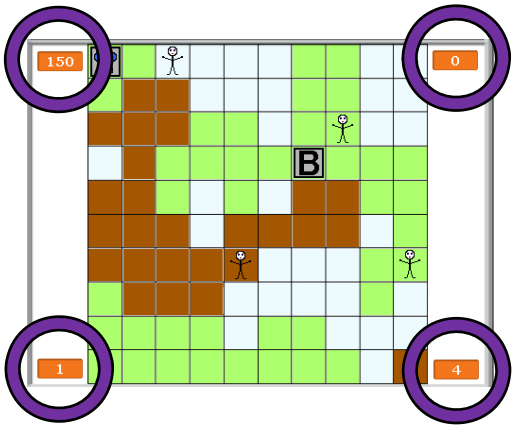
2.5



2.6

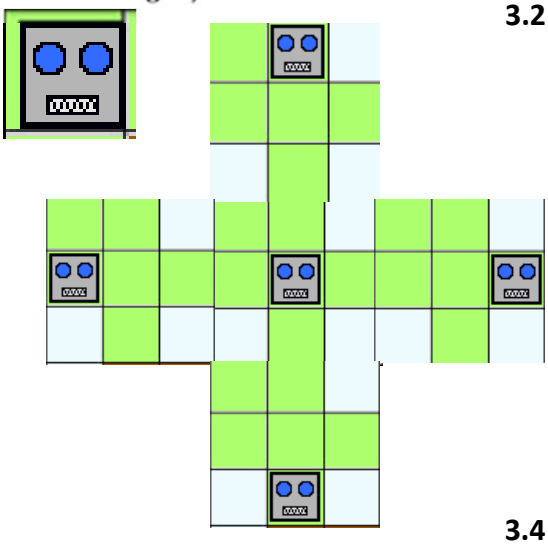


2.7

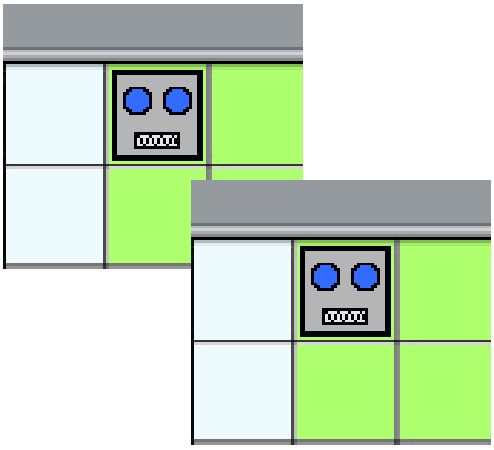


3.1

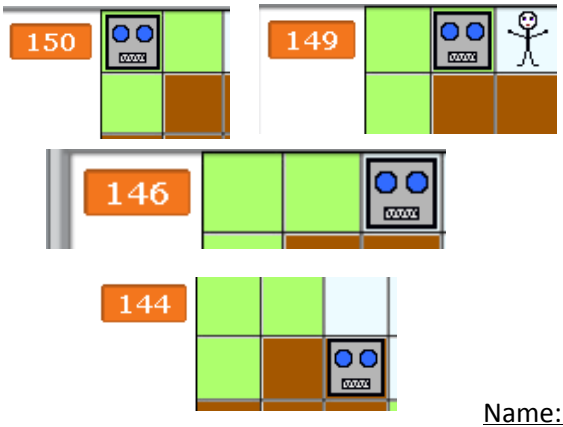
(Before Changes)



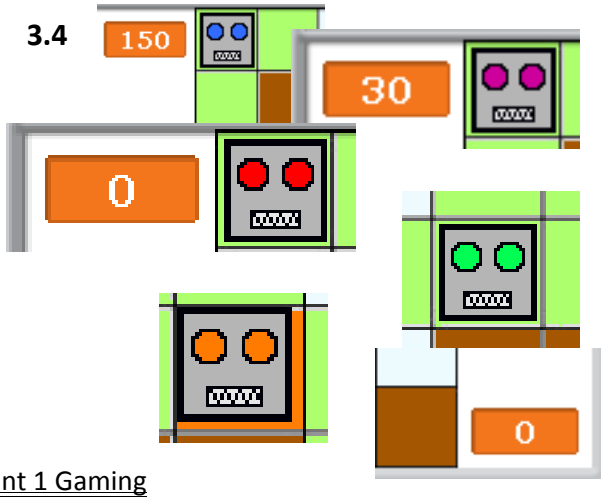
3.2



3.3



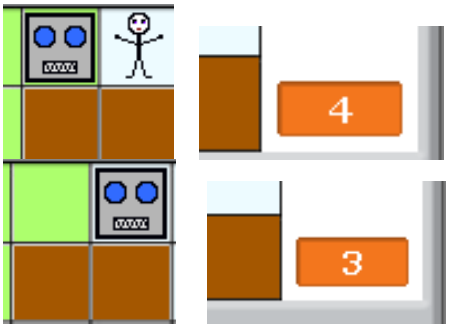
3.4



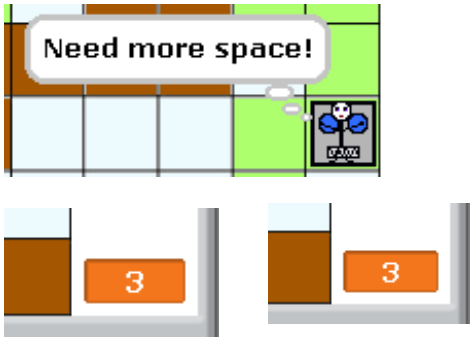
Name: Student 1 Gaming

Candidate Number: 0000

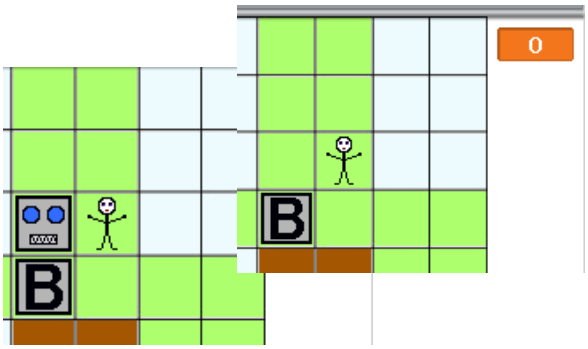
3.5



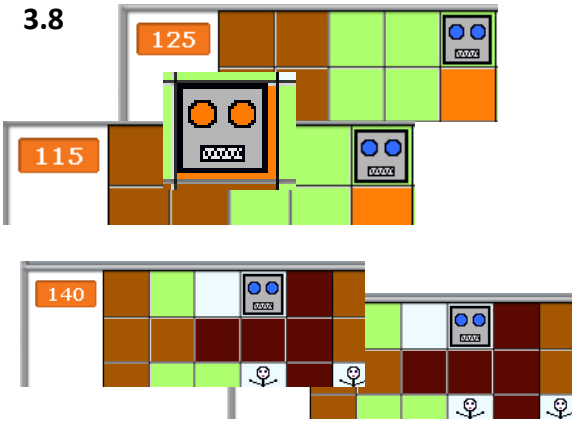
3.6



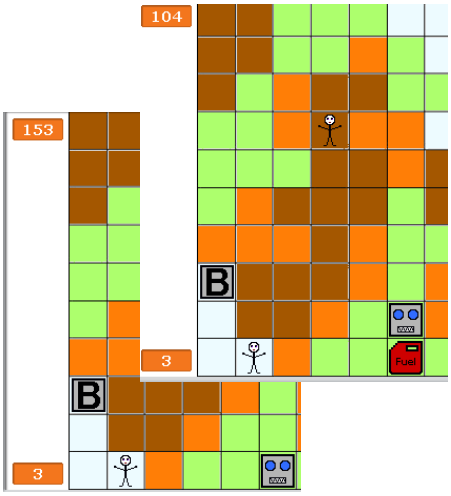
3.7



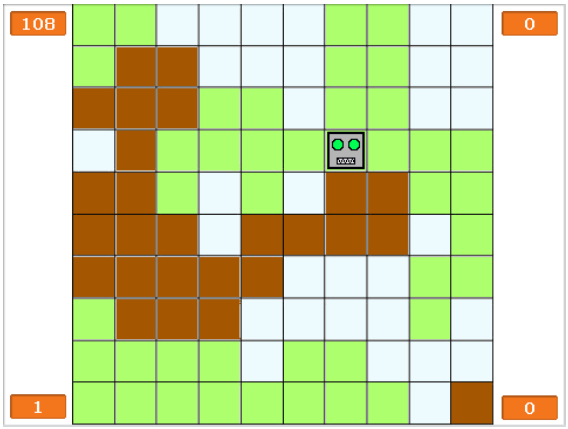
3.8



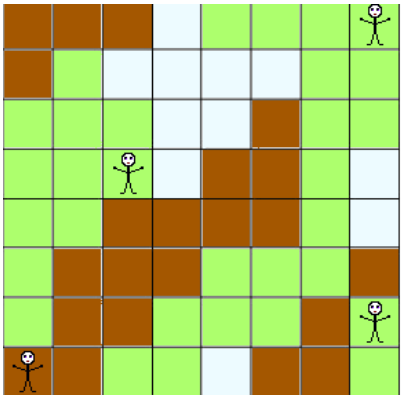
3.9



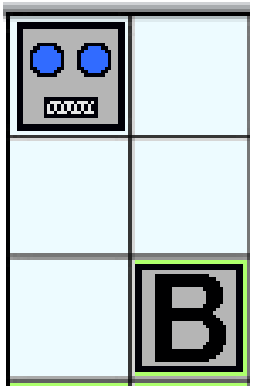
4.1



4.2



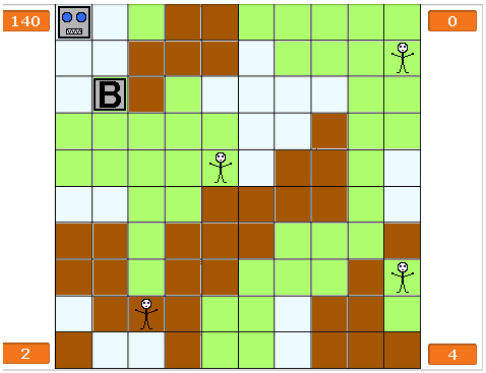
4.3



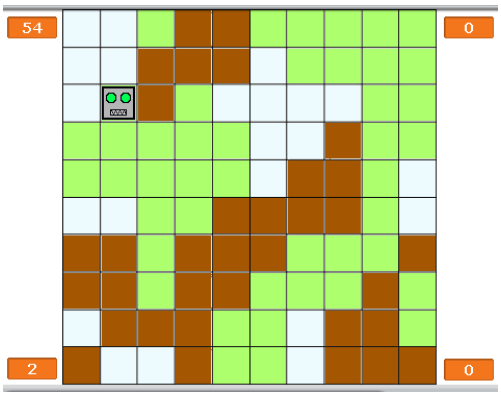
Name: Student 1 Gaming

Candidate Number: 0000

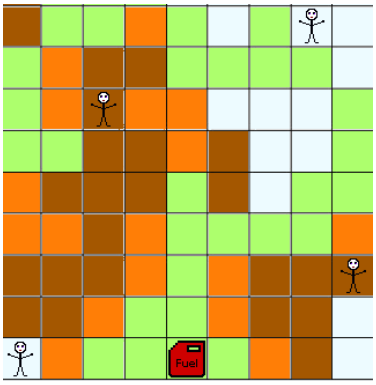
4.4



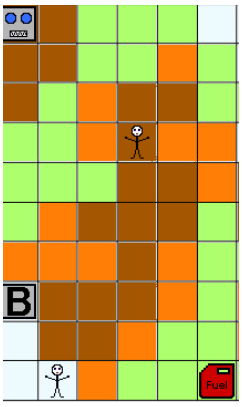
5.1



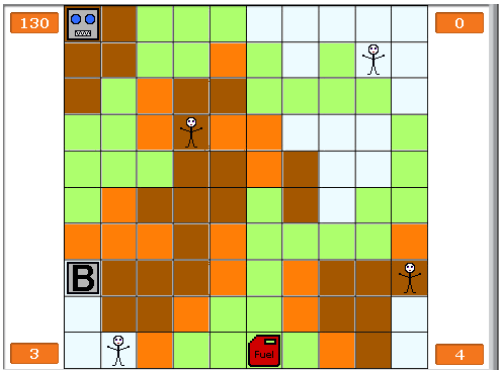
5.2



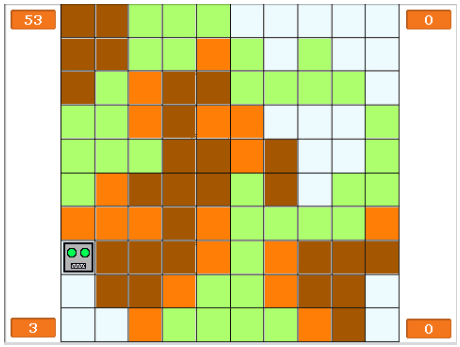
5.3



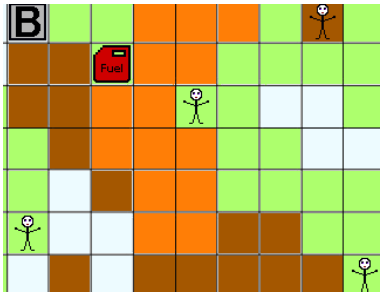
5.4



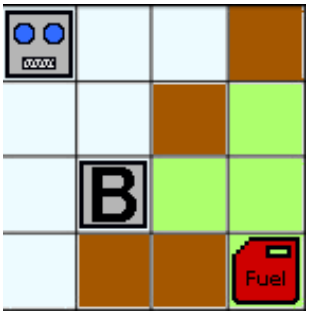
6.1



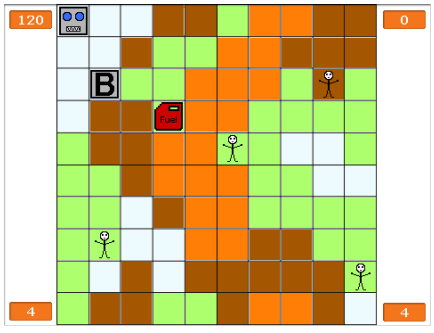
6.2



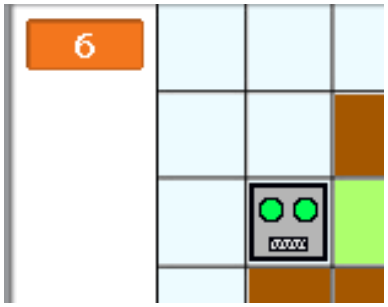
6.3



6.4



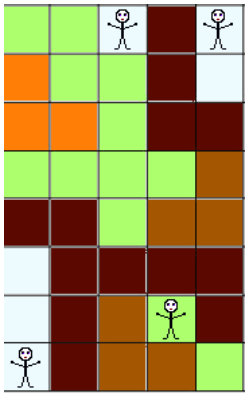
7.1



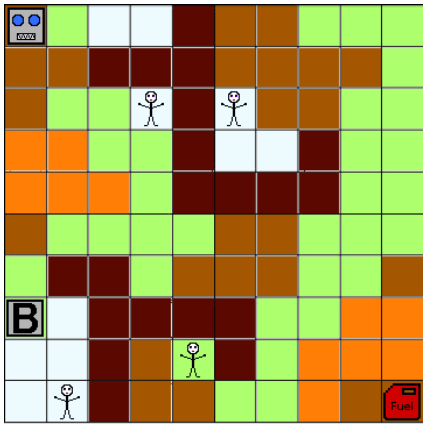
Name: Student 1 Gaming

Candidate Number: 0000

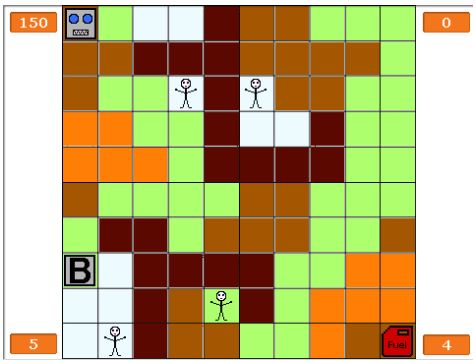
7.2



7.3

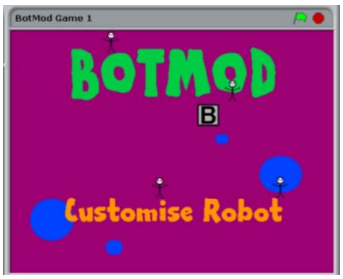


7.4

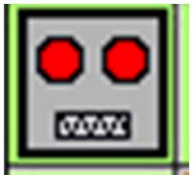


8.1

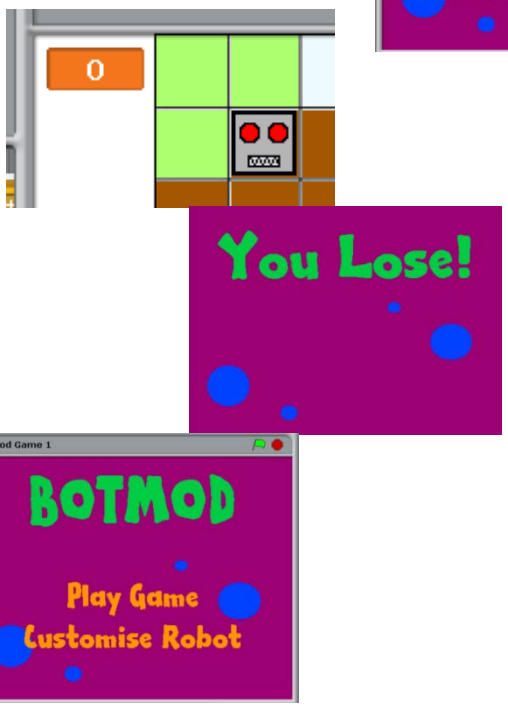
(Before Changes)



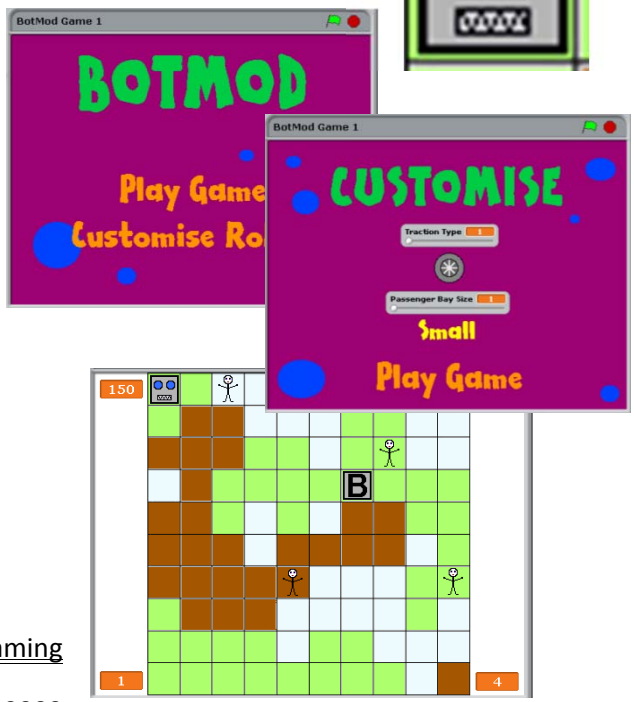
(Before Changes)



8.2



8.3



Name: Student 1 Gaming

Candidate Number: 0000

Data Structures.

Structure	What It Does	What Does It Affect	Why Is It Used
Sequence – Green Flag	Starts the game, allowing all other code to happen.	It starts the gameplay itself, along with all other code in the program, such as variables that are declared at the beginning of the game.	To start the game at the Main Menu, initiating all other code as well as allowing the user to play.
Selection – If	Checks the current terrain's colour.	The Power Units used every time the Robot moves, which will vary dependant on the terrain the Robot is currently on.	As the terrain creates variety in the gameplay for the user to be tactical about where the Robot should move.
Selection – If	Checks which traction type & passenger bay options have been selected by user.	The number of Power Units used when the Robot moves, also taking the terrain into consideration.	Adds the element of strategy into the gameplay, giving the user a decision to make as to what modifications the robot requires.
Sub Routine – (As Broadcast)	When started, the broadcast will check what colour the current terrain the Robot is on, before deducting this in the form of Power Units.	It affects the Power Units that get used by the Robot, as well as the costume (only if the terrain in question is Lava). The Power Usage variable is determined by the terrain and traction type, and is then subtracted from the Power Units themselves.	It occurs as a result of the Robot's movement, meaning every time the Robot moves a square, this same piece of code is called up.
Iteration – Forever If	Constantly checks whether the Power Units are below 31 or if they equal 0, resulting in costume changes.	The Robot costume changes as a result of having low (purple eyes) or no (red eyes) fuel. The level will also return to the Main Menu after a Losing Screen as the player will have lost.	This code checks whether the play has ran out of fuel, the ability to lose being a necessary element to the game, making the player cautious over what moves they make.
Iteration – Forever If	Checks if an arrow key has been pressed, changing the Robot's position as a result.	It affects the x and y coordinates of the Robot itself, as well as the Power Units that will be used as a result of the Robot moving.	It is used as otherwise the user would be unable to interact with the Robot completely during the levels themselves.
Iteration – Forever If	Waits until the Robot makes contact with the Passenger (occurs separately for all 4 passengers in the game)	It affects variables regarding the amount of Passengers left on the grid, as well as the number of Passengers in Bay. The Passenger will also disappear if the Robot has space to do so.	It is a vital part of the game, very similar to the other Forever If related to contact with the Base, which changes the current level of the game.

Name: Student 1 Gaming

Candidate Number: 0000

Data Types.

Variable	Data Type	Example
Arrow Key	Character	U
Level	Integer	5
Moveable	Boolean	0 or 1 (True/False)
Passenger Bay Size	Integer	3
Passengers In Bay	Integer	1
Passengers Left	Integer	4
Power Units	Integer	150
Power Usage	Integer	5
Traction Type	Integer	3

Scratch's data types consist only of Integer, String, Real and Boolean (although the Arrow Key variable is only ever set to U, L, R, or D). The Moveable variable is also basically an integer value, although it is only ever set to 0 or 1, to determine whether the robot is able to move or not.

There are no constants in a Scratch game, as all values need to change, therefore they will be variables. Variables are automatically assigned their data type by Scratch (for instance, if a number is 'said' then it becomes a string, but will be an integer/real number if used in calculations).

Scratch programs do not support arrays, although in this case none were needed (and could've been averted if this wasn't the case).

Efficiency & Robustness.

1) Efficiency:

- If code is efficient, it deals with the necessary programming in as little space with as little time involved as it can, without becoming easy to break. Efficient code is very useful in a program to prevent losing track of where certain code takes place and not only speeds up the development of the game, but often prevents loose ends in the code appearing, that can then lead to unnecessary bugs and glitches in the game.
- Much of my code was dealt with efficiently through the use of variables, an example being my 'Moveable' variable. As opposed to having to check whether the robot is on the border of the grid, or whether the game is still at the Menu screen, etc. the code allows this one variable to be checked over and over, preventing unnecessary and long code to check if random and separate conditions are met. This will also make the code more ***robust***.
- Another key component was the use of broadcasts and the repetition of certain code over and over. For instance, the progression from one level to the next is done using the same pieces of code that are used over and over and called upon every time the robot has landed on the base with no passengers being left on the grid.

2) Robustness:

- If code is robust, it is tough to break, glitches and bugs are hardly ever/never encountered, and every eventuality that could arise is covered within if statements and the code as a whole.
- I ensured my code was robust through both rigorous testing and proof-through my code, both tricks helping me to fix certain bugs, one or two of which were major issues. One in particular was the issue with replaying the game after the user has lost once already, as the robot's eyes would still appear red. This was because there was no code telling it otherwise, and this eventuality had not been dealt with by the code. It was easily fixed and thanks to this testing other issues with encountering the lava and wall squares in my later levels were also fixed.

Game Evaluation.

1) User Requirements:

- My game has met all of the set specifications necessary to be included in the game. There is a robot, who picks up passengers and loads them into a bay. There is both a Main Menu and Customisation Menu, in which the user can start the game and modify the aspects of the robot. The robot has a set number of power units, as well as different types of Traction and Passenger Bay Sizes to choose from. There are in total five levels, with an increase in difficulty on each one, and all of this takes place on 10x10 square grids.
- Not only have all the specifications been met, but the game also includes additional features to improve the overall gameplay.
    - For instance, there are a total of five levels, all of which get more difficult.
    - I created a Lava square, used from level three onwards to add both difficulty and variety to the experience, as well as making it easier for Tracks (as they're the only modification that an drive through Lava, countering the disadvantage that Tracks have in that they use up a full three power units no matter what the terrain).
    - I also added a Wall square, used only in Level 5, as otherwise the gameplay would become repetitive, especially considering how Level 5 is the final level in the game.
    - To prevent the game from being impossible, I made a Fuel Can sprite which, in Level 3 and above, appears on the grid, giving the robot an additional sum of power units (using the formula "50(Level - 2)" only on Level 3 and above). The Fuel Can on Level 5 is positioned in the corner, trapped by Lava, as the final level can in fact be completed without it being needed (unless using Tracks, which can drive through Lava).

2) The Menu Screens:

- The Main Menu simply contains the game title, with both a 'Play Game' and 'Customise' button at the bottom. When clicked, the Play Game sprite will trigger a broadcast that will start the game. However, the user will probably opt for clicking the Customise sprite first, in order to change the Robot's parts.
- The Customising Menu displays two variables that are sliders, with a minimum setting of 1 and a maximum of 3. This means that the user can change these variables, which will change the sprite's costume positioned below the variable. The Play Game sprite is positioned at the very bottom of the screen on this Menu, and has exactly the same function that it has on the Main Menu itself.

- I think the Menus' design works well with the game, in that it's both simple (therefore suited to a younger audience) as well as containing more stylish purple colour, etc. The fonts have a playful impression about them and the slider variables create more fun out of the user interaction within the Customising Menu than normal sprites would, therefore adding to the entertainment value of the game.

3) The Levels:

- Level 1
  - The first level kicks off the gameplay and as instructed, the Robot is able to move within the grid (every move using a specific number of Power Units depending on the Traction Type, Passenger Bay Size and Terrain), while not being able to move off the grid in any direction. The Robot's movement is precise and aligned to the grid, and so are the positions of the Passengers and Base.
  - With sufficient space in the Passenger Bay, the Robot can pick up a Passenger by moving to a space in the grid occupied by the Passenger. The Passenger disappears, and the variables change accordingly. When contact with the Base (through the same method) occurs, the Passengers are removed from the Bay.
  - The game runs perfectly well, with virtually no glitches whatsoever, although I have 2 small issues:
    - Firstly, the grid itself was extremely difficult to create. I used Word, 2D Design and Serif DrawPlus, however all methods had a similar problem. On Scratch they would be just shy of the intended dimensions they were supposed to be, therefore the Robot's movements had to be adjusted to 35.9 steps as opposed to 36. Although the end result was fine, with no hint toward this small adjustment being in place, it was more time consuming than expected to create this element of the game.
    - Secondly, the variables (although displayed and fully functional at all points in the game) did not have labels alongside them. To make them fit on the sides of the screen, I turned their respective displays to 'Large Readout', meaning the variable's name was not displayed with the value itself. I was planning to fix this, however as the game progresses the user quickly seems to gain an understanding of which variable's which, so I decided not to waste time on a trivial matter such as labelling the variables (for instance, it is clear the variable that starts on 150 and decreases with every movement is some form of Power Unit value).
- Level 2

- The progression to Level 2 is fairly easy, and is possible to achieve using any combination of the Robot's modifications. Level 2 has a different background, and additionally the Passengers and Base all move to new positions, adding variety to the game, as otherwise the user could easily get bored, considering that other elements of the game (such as Scratch's rather limited graphics do little to add to the overall enjoyment of the game.
- Every aspect of this level works as it should, and the Power Units start at 140, increasing the difficulty of the game. This was done as instructed, although it does result in some combinations (such as Tracks and a Large Passenger Bay) making the completion of the level impossible. This means that when the user returns to the Main Menu, they will have to change the customisations made to the Robot.

- Level 3
  - This level adds many new elements to the gameplay, as otherwise the gameplay would become dull and rather boring, considering the player would just be doing the same actions on the same terrain over and over. The user starts with 130 Power Units this time, again adding to the difficulty, but the main change is the addition of Lava.
  - I added Lava for added variety and difficulty, as well as making the Tracks option a little easier to use from this point in the game onwards, as using the Tracks, the Robot can safely step onto a Lava square without any problem. Any other combination of modifications to the Robot result in the Robot's eyes temporarily turning orange, along with the Robot being returned to its original spot before the contact, accompanied by a 10 point Power Unit deduction. I think the Lava square was a fantastic decision, as the game's levels would otherwise become increasingly repetitive.
  - Another added feature at this point in the game is the Fuel Can sprite. This granted the Robot extra fuel to help complete the level, when the Robot moves to the same square. The Power Units are changed using the formula "50(Level No. - 2)" as otherwise the later levels become increasingly difficult and eventually impossible no matter what the combination. I think the addition of this sprite was crucial as, using the table of Power Unit deductions provided, the game was impossible for any modifications on the Robot. The sprite itself makes its aim very clear and means that there doesn't have to be any form of explanation to the user, as the sprite is self-explanatory.
- Level 4
  - This level yet again utilises the Lava square and exploits its purpose as much as possible. The layout of the background means that the Robot has to travel to the bottom of the grid to get to the right half of the level, which is where most of the Passengers reside. This results in the user needing to conserve as many Power Units as possible in order to complete the level, even taking into

account the Fuel Can's vital addition of 100 Power Units. In terms of level design, I'm very pleased with the decision to create Level 4 this way, as again it adds plenty of variety to the gameplay compared to having the same dull layout with the same 3 Terrain squares.

- ❖ This level is a challenge, and eliminates many Robot combinations from completing the game at all, but that cannot be helped without changing the amount of Power Units lost per move using Tracks or Skis. The Power Units start at only 120 on this level.

- ➢ Level 5
  - ❖ The 5th and final level adds one final Terrain square to the mix. A Wall. The code was relatively easy and was therefore a worthy decision, as the variety in this level reaches its peak – any other additions and the gameplay would've started to become a little too complex considering the game's low-key graphics.
  - ❖ The Power Units (in order to make the completion of the game possible at all) is reset to 150, as I made a clever move with the Fuel Can. If the user has somehow made it this far using the Tracks, then they are able to travel through Lava and access the Fuel Can, as without it the level would be impossible, however using Wheels or Skis the user is actually able to complete this level without need for the Fuel Can, therefore to make it accessible would be a stupid decision, making the final level too easy.

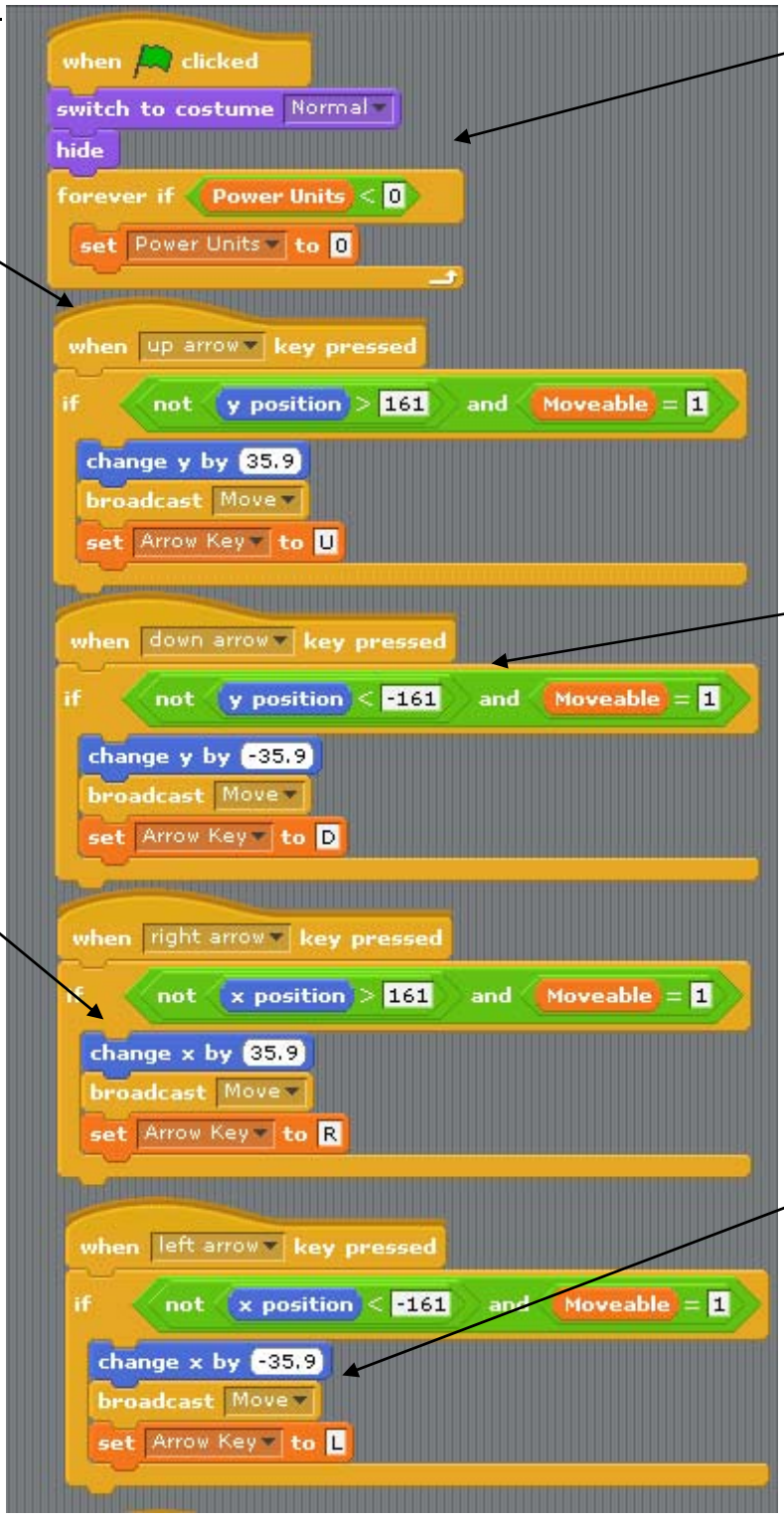
4) Winning & Losing:

- ➢ When the game is won, the Robot's eyes yet again turn green, to signal the completion of the level, before the background changes to a screen with similar aesthetics to the Main Menu, with the simple heading of 'You Win!' at the top. The game then resets, returning the user to the Main Menu, giving the player the ability to play the game again. The only criticism I have to the completion of the game is the lack of sound perhaps, as the addition of a small, upbeat sound file for instance would highlight the fact that the user has beat the game a little better than just a screen telling them this. As with all arcade games, little sounds like this really add to the entertainment value of the game, as without these features, the game doesn't appear to be as worthwhile when one completes the game.
- ➢ When the game is lost, however, a different screen will appear, simply headlined 'You Lose!' before again, the Main Menu is displayed. The user then has the chance to play again, as well as to re-customise the Robot itself. This adds an element of tactics behind the customisation of the Robot, as the player eventually learns which combinations are more effective. However again the addition of sound or some form of animation would benefit the game's entertainment value.

The Final Game.

1) The Robot:

Up, Down, Left & Right Arrow Key code.



This stops the Power Units running below zero when a move would cause this to happen. Just before this, the costume is clearly set to Normal at the beginning of the game.

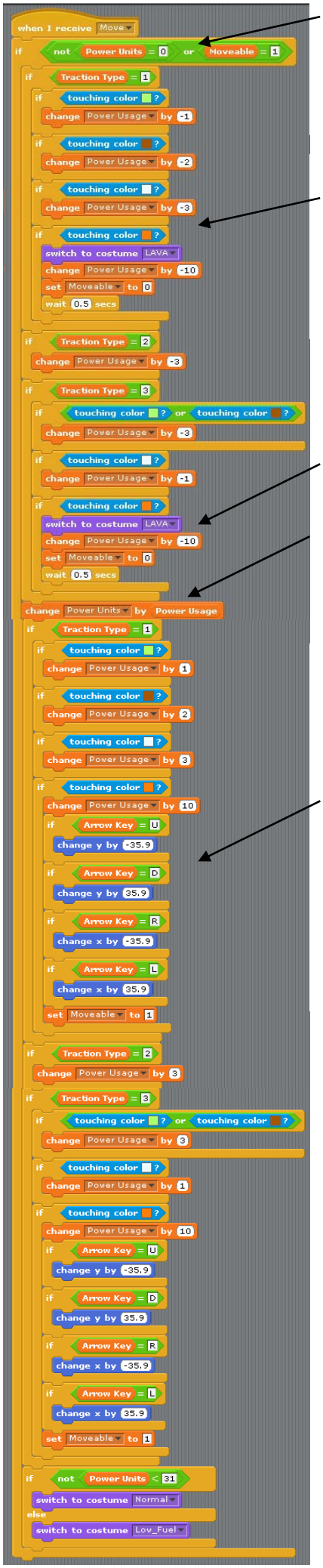
Each one checks the x/y position of the Robot, depending on whether a horizontal or vertical motion is to be carried out.

The Robot will then be moved 35.9 steps in whatever direction is required, before the Move broadcast occurs.

The Arrow Key variable is also changed at this point.

Name: Student 1 Gaming

Candidate Number: 0000



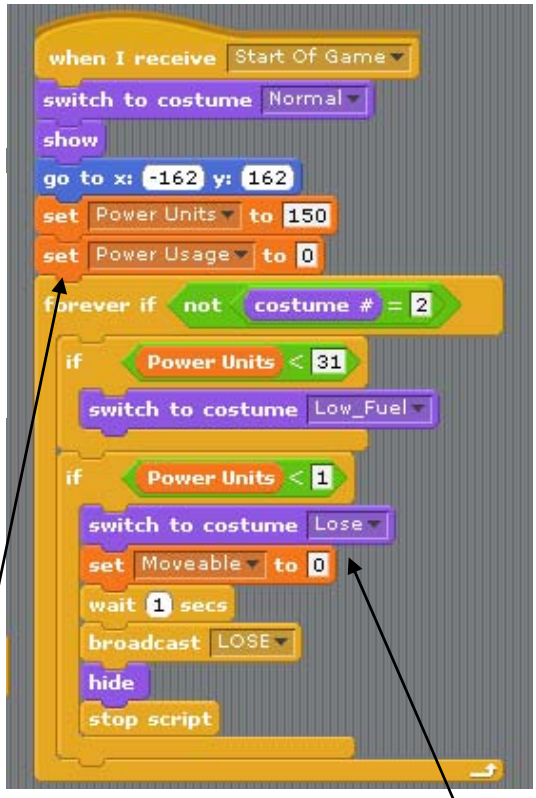
This all occurs when the Robot moves, and both the Power Units available are sufficient and the Robot is able to move.

If the Wheels are in use, then depending on the Terrain, the Power Usage variable will be adjusted. This is then repeated for other Terrains for both Tracks and Wheels.

When contact with Lava is made, a costume change and delay will occur in addition.

The Power Units are then deducted from the Power Usage.

The Power Usage is then reset to what it was before (this code is needed as opposed to just 'setting Power Usage to 0', as the Passenger Bay Size uses the same variable for efficiency).



When the broadcast is received (when the actual gameplay begins), the Robot is set to its opening state and position.

The following code checks the Power Units variable. If low, the Robot will appear with purple eyes. If 0, the user will have lost the game.

Similar to the movement of the Robot, this code is executed the moment the Robot makes contact with a Wall. He will move backward opposite to the direction in which he moved first.

When the Base has been reached, causing a level change, the following will happen. The Robot's eyes will turn green, and he will return to his original position and state, after resetting some of the variables.

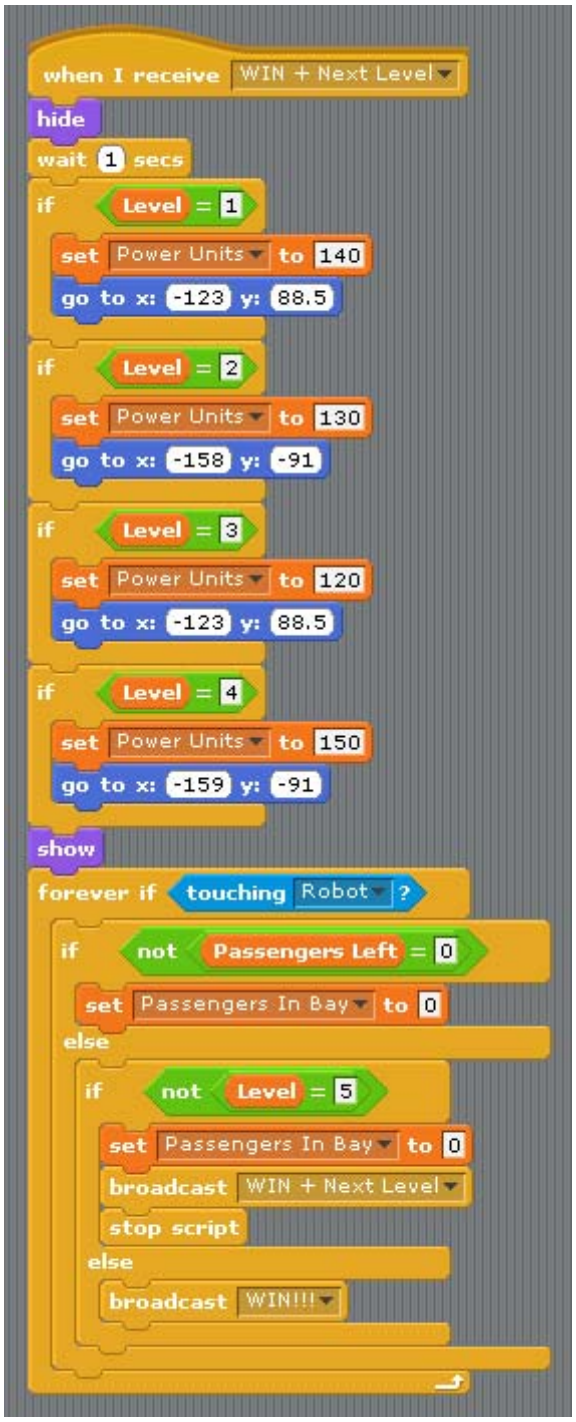
The code below (called upon victory) will hide the Robot and return the Robot to the start.

Name: Student 1 Gaming

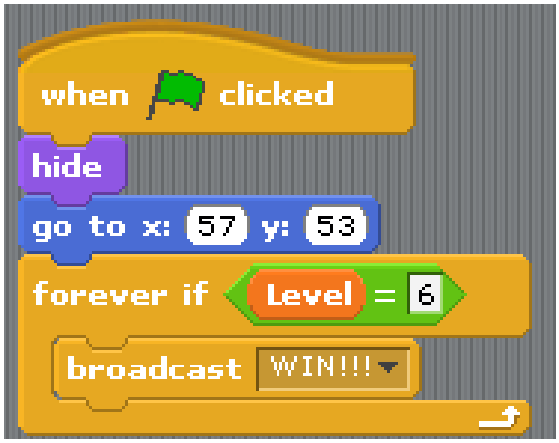
Candidate Number: 0000

2) The Base:

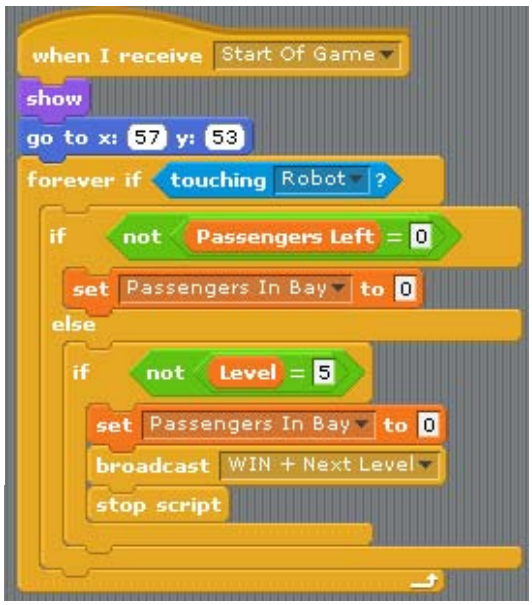
Depending on the level, the Base will change position as well as adjust the amount of Power Units the Robot is offered. The Broadcast at the bottom is used when the Robot makes contact with the Base, assuming all conditions are met.



The final broadcast will be made for when Level 5 has been completed, launching the Win procedure. The Base itself will respond to this Broadcast by hiding and returning to its original place.



This code on the right is identical to that on the left, as the left code only occurs after each new level, whereas it is also needed for the start of the game as well.



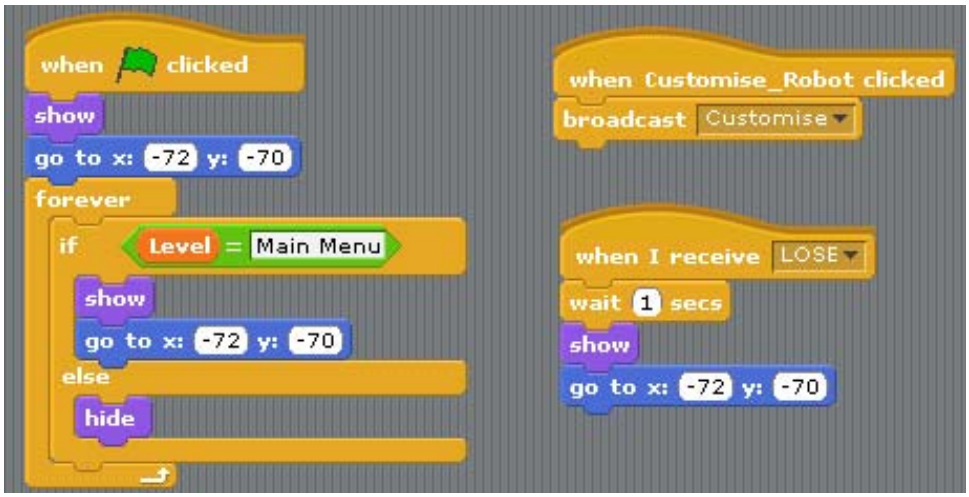
3) Play_Game:

At the beginning of the game, this sprite will move dependant on the Main Menu screen in use at any given time.

When clicked, the Start of Game Broadcast will occur, starting the actual gameplay. The Power Usage variable is also adjusted to whatever had been selected as the Passenger Bay Size.

A repeat of the code at the bottom right that will only occur at the very beginning of the game, as the one on the right will occur multiple times whenever the user progresses to the next level.

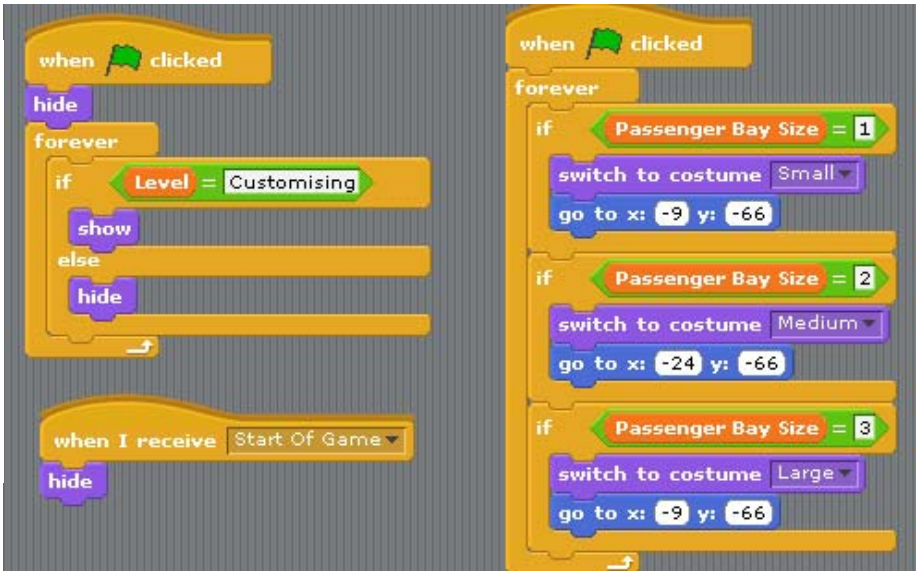
4) Customise_Robot:



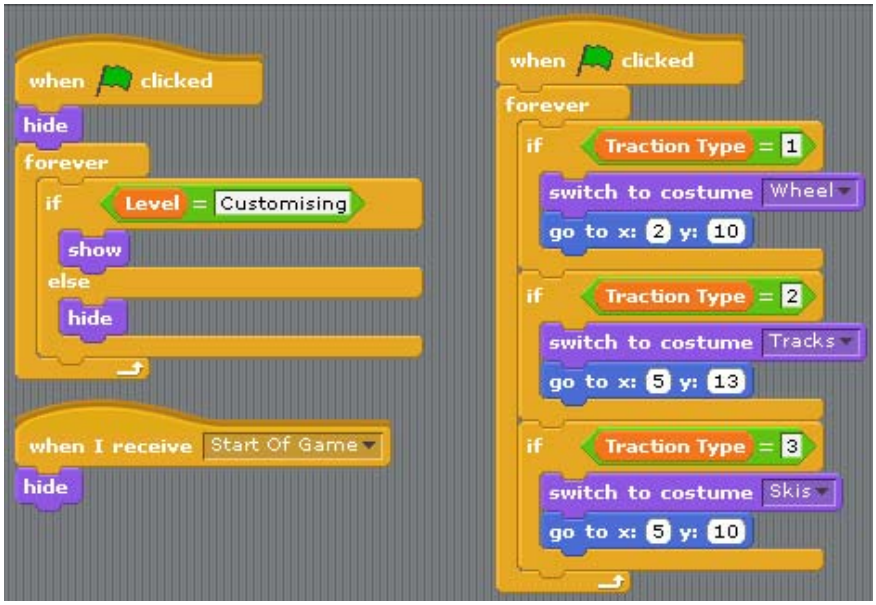
The left code is similar to that of the Play_Game sprite, meaning that it's position (or whether it is to be displayed or not) will be decided upon a forever if loop, asking what Menu is in use at the given moment. The Menu screen will change upon clicking this sprite, and it will reappear when the user has lost the game.

5) S/M/L:

If the Customising screen is in use, then this sprite will show, hiding in any other case. The Passenger Bay Size will change when the user adjusts the slider variable on-screen, leaving an image of the words 'Small', 'Medium' or 'Large' as a result of this change.



6) Traction_Type:



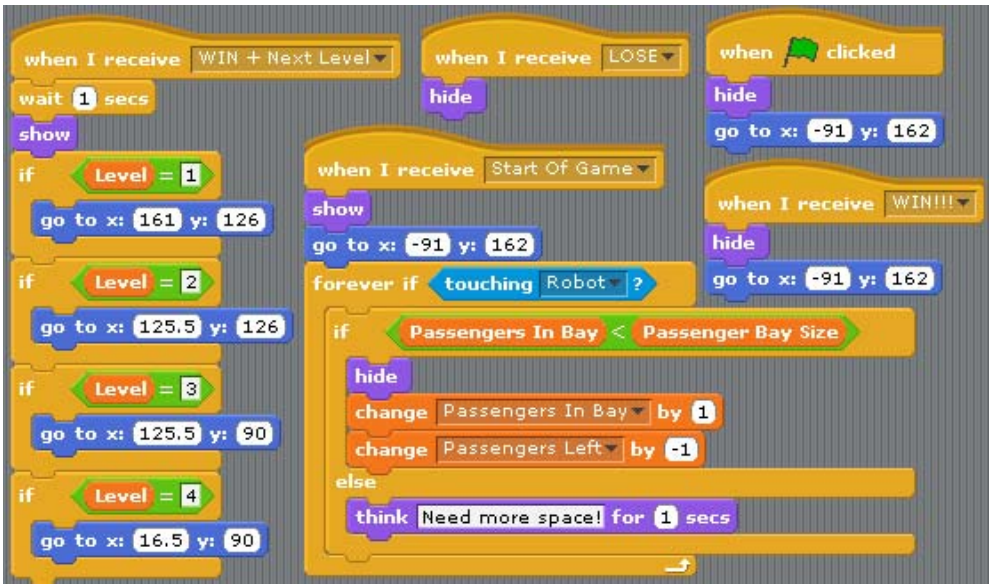
Much like the code above, this sprite only comes into play on the Customising screen. Additionally, the sprite's costume will change dependant on the other slider variable, the Traction Type. This will yield results of either a Wheel, Tracks, or Skis, to indicate what choice the user has made.

Name: Student 1 Gaming

Candidate Number: 0000

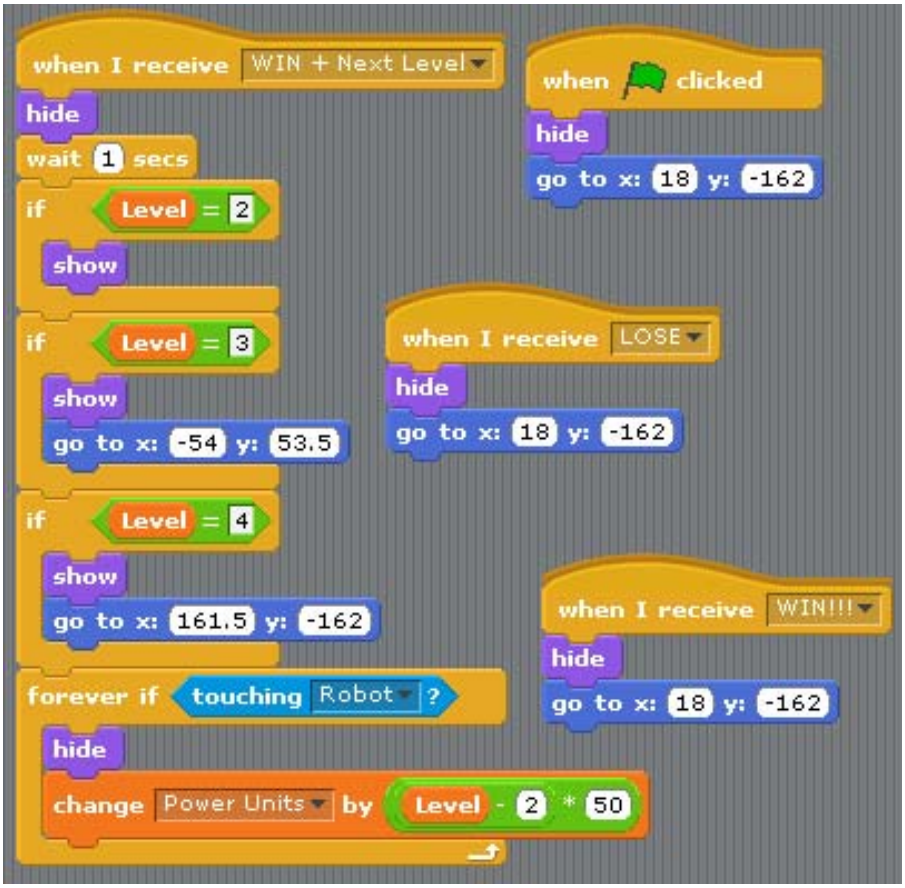
7) Person_1 (Persons_2, 3, & 4 all nearly identical):

Upon progression to the next level, the Person will change position (the individual positions are different for each Person sprite). They will all hide when the game has been won or lost, and will also hide upon the beginning of the game, being placed in the starting positions at this point also. If they make contact with the Robot then, providing the Robot has space in its Passenger Bay, the sprite will hide, and be removed from the grid, the variables indicating this change.



8) Fuel Can:

The Fuel Can sprite will remain hidden until the 3rd level, where it will appear at the position designated by the code at the top-right. This position will also be set (as well as the sprite hiding) when the game is won or lost.



As indicated by the code at the very bottom-left, the amount of Power Units offered to the Robot, when contact is made with the Fuel Can sprite, will be determined by the Level number, in the expression '(Level-2)*50'. So, for example, when the level is 4, the expression will return 2*50, meaning 100 Power Units will be given to the Robot.

9) Stage:

The Stage mostly controls the variables, particularly at the start of the game. All variables used in the game are at first hidden, as the Main Menu is the first aspect the user experiences at the start of the game. The Customisation screen reveals the two slider variables, and the game itself will hide those and show the 4 on-screen variables used during gameplay.

The Level (as displayed at the bottom-right) will change the background, and at the bottom-left, the code the winning the game is in use. All 4 variables used during gameplay are hidden and the Level is reset, while when the game is lost, a similar thing will happen. The Level variable also changes upon every 'WIN + Next Level' Broadcast.