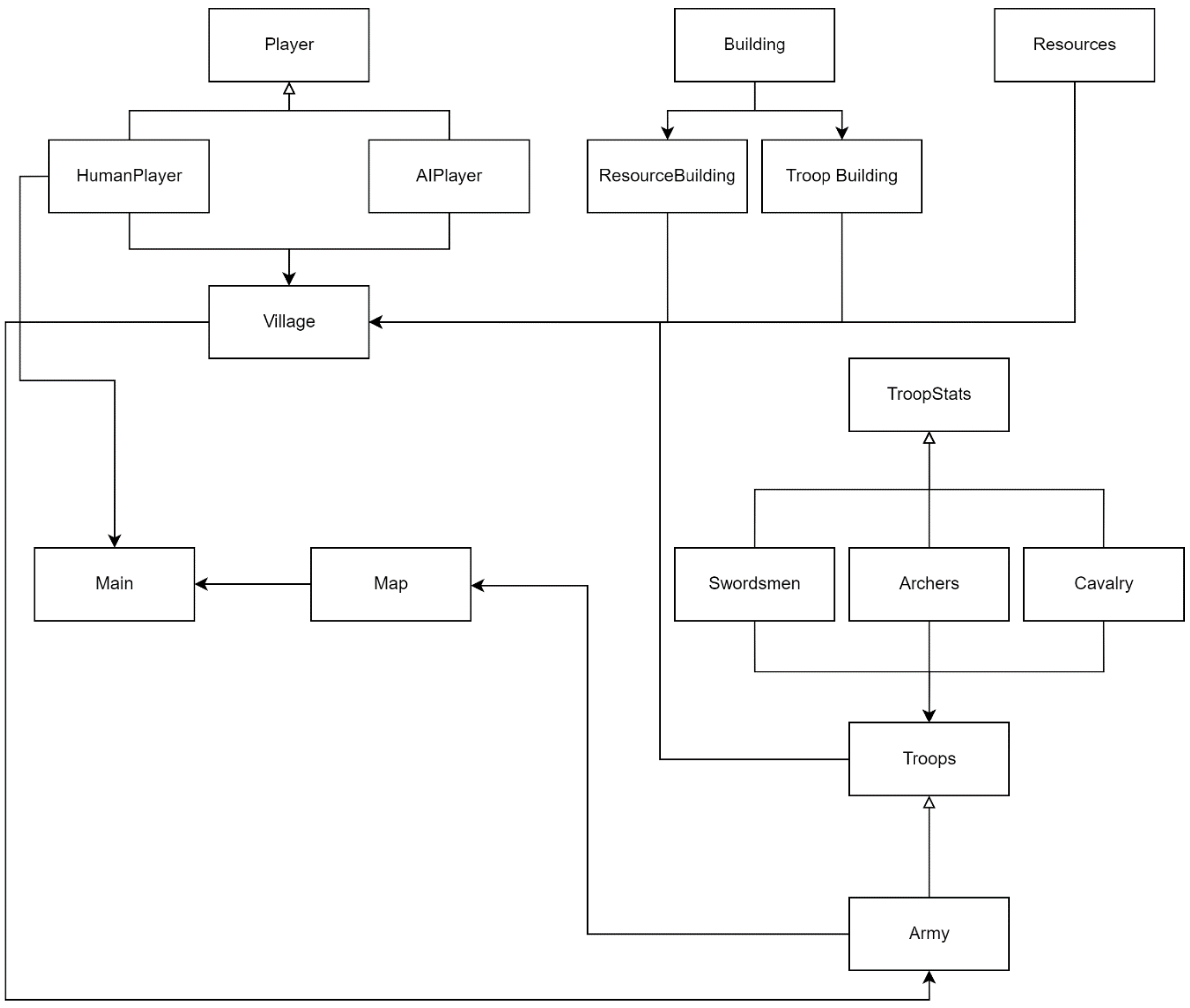
Repository URL: https://github.com/kelsey-um/cps2004-assignment

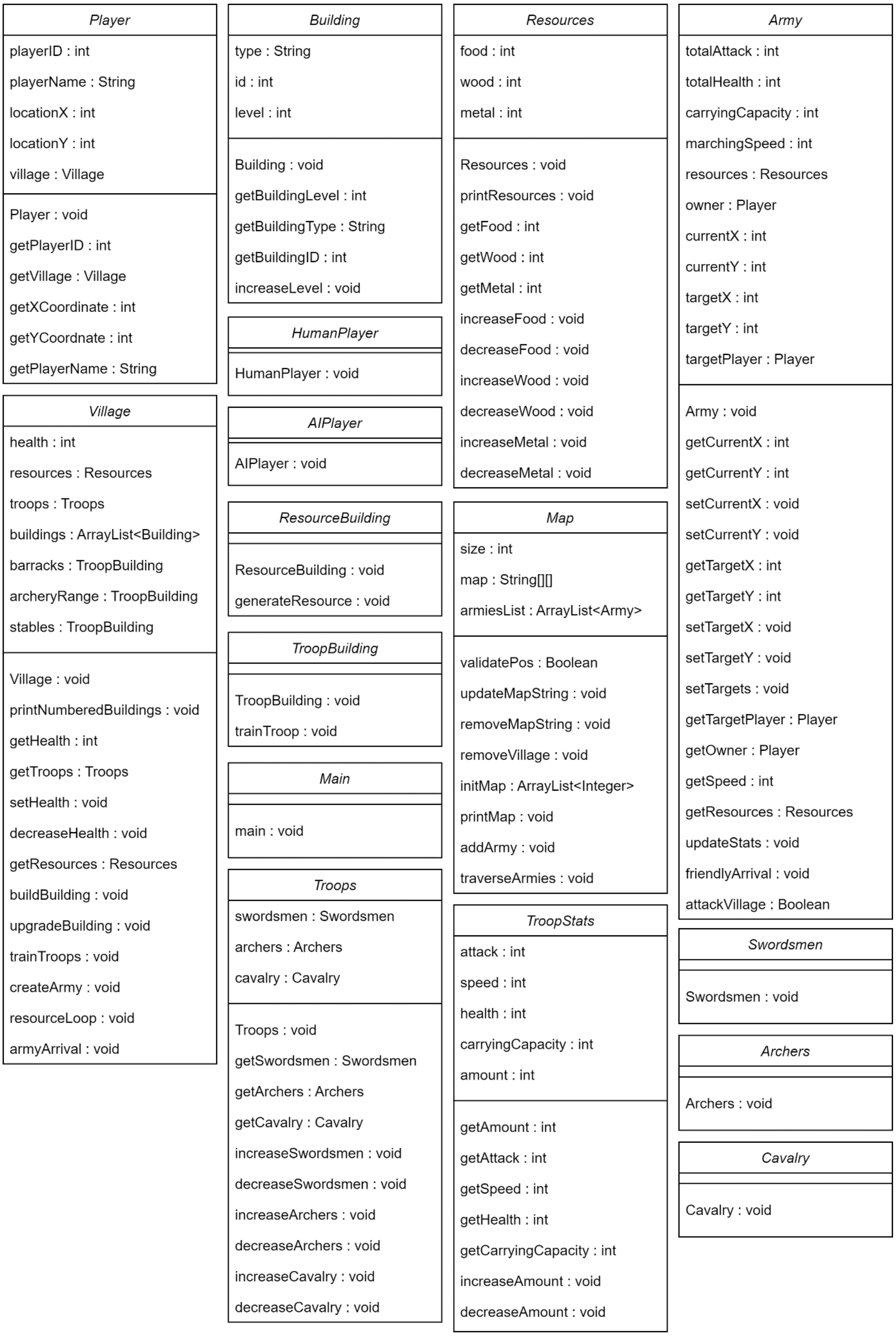
Last Commit Hash: bc138bb1100f2399bfeeaeffa710c38935afb969

Task 1 – Village War Game

UML Diagram

Due to the size of the UML diagram for this problem, the diagram was split into two. The first diagram is a simplified version where only the names of the classes and how they interact with each other are shown. The second diagram has the variables and methods for each class listed.





Player
playerID : int
playerName : String
locationX : int
locationY : int
village : Village

Player : void
getPlayerID : int
getVillage : Village
getXCoordinate : int
getYCoordnate : int
getPlayerName : String

Village
health : int
resources : Resources
troops : Troops
buildings : ArrayList<Building>
barracks : TroopBuilding
archeryRange : TroopBuilding
stables : TroopBuilding

Village : void
printNumberedBuildings : void
getHealth : int
getTroops : Troops
setHealth : void
decreaseHealth : void
getResources : Resources
buildBuilding : void
upgradeBuilding : void
trainTroops : void
createArmy : void
resourceLoop : void
armyArrival : void

Building
type : String
id : int
level : int

Building : void
getBuildingLevel : int
getBuildingType : String
getBuildingID : int
increaseLevel : void

HumanPlayer

HumanPlayer : void

AIPlayer

AIPlayer : void

ResourceBuilding

ResourceBuilding : void
generateResource : void

TroopBuilding

TroopBuilding : void
trainTroop : void

Main
---
main : void

Troops
swordsmen : Swordsmen
archers : Archers
cavalry : Cavalry

Troops : void
getSwordsmen : Swordsmen
getArchers : Archers
getCavalry : Cavalry
increaseSwordsmen : void
decreaseSwordsmen : void
increaseArchers : void
decreaseArchers : void
increaseCavalry : void
decreaseCavalry : void

Resources
food : int
wood : int
metal : int

Resources : void
printResources : void
getFood : int
getWood : int
getMetal : int
increaseFood : void
decreaseFood : void
increaseWood : void
decreaseWood : void
increaseMetal : void
decreaseMetal : void

Map
size : int
map : String[][]
armiesList : ArrayList<Army>
---
validatePos : Boolean
updateMapString : void
removeMapString : void
removeVillage : void
initMap : ArrayList<Integer>
printMap : void
addArmy : void
traverseArmies : void

TroopStats
attack : int
speed : int
health : int
carryingCapacity : int
amount : int
---
getAmount : int
getAttack : int
getSpeed : int
getHealth : int
getCarryingCapacity : int
increaseAmount : void
decreaseAmount : void

Army
totalAttack : int
totalHealth : int
carryingCapacity : int
marchingSpeed : int
resources : Resources
owner : Player
currentX : int
currentY : int
targetX : int
targetY : int
targetPlayer : Player

Army : void
getCurrentX : int
getCurrentY : int
setCurrentX : void
setCurrentY : void
getTargetX : int
getTargetY : int
setTargetX : void
setTargetY : void
setTargets : void
getTargetPlayer : Player
getOwner : Player
getSpeed : int
getResources : Resources
updateStats : void
friendlyArrival : void
attackVillage : Boolean

Swordsmen

Swordsmen : void

Archers

Archers : void

Cavalry

Cavalry : void

Design and Implementation

The Village War Game is split into multiple different classes as can be seen in the diagrams above.

Main

This class contains the main method which executes when the program is run. The declaration of all the players can be found here. The game loop can also be found in this method.

Map

This class is responsible for the management of the map of the game.

- validatePos
  This private method takes in two coordinates, x and y, and the size of the map and checks if they are still within valid range. This is to make sure the IndexOutOfBoundsException error is not encountered when accessing the map array.
- updateMapString
  This method is used to update the values in each cell of the map when the army is traversing on the map.
- removeMapString
  This method is used to remove the army off the map after it has arrived at its destination.
- removeVillage
  This method is used to remove the village from the map after it has been destroyed.
- intiMap
  This method is used to place all the villages on the map according to the amount of players that will be playing. The coordinates of the villages are randomly generated. The only condition that needs to be met is that there cannot be a village in another village's adjacent cells. This was done so the villages can be more spread out on the map. The method returns an ArrayList of all the coordinates which is then used in the main method to assign each Player their village coordinates.
- printMap
  This method is used to display the map to the user.
- addArmy
  This method is used to add an army to the armiesList after it is created.
- traverseArmies
  This method is used to move any armies in the armiesList. The armies first move along the x-axis and then the y-axis until they arrive at their destination. There is no diagonal movement. In this method, there is also where the result of an attack is handled (i.e., deleting the army when it is defeated or returning it home when successful).

Player

This class contains the Player and its two inherited classes HumanPlayer and AIPlayer. Due to the AI not being implemented, these two inherited classes do not have any unique methods or variables to them. When a new Player is declared, all the variables are assigned in the constructor and throughout the game these do not change. Therefore, the only methods found in this class are get methods to be able to access these variables.

Village

This class contains the Village. There are a number of get and set methods to be able to modify the variables. There are also the following methods:

- printNumberedBuildings
  This method is used to print out all the buildings found in the ArrayList buildings. This is used to show the user the buildings in the village along with a number in the front.
- buildBuilding
  This method is used to build a new building in the village. Through the use of loops and switch cases, the user can build any building and the cost for it is reduced from the village's resources. As part of the game rules, there can only be one barracks, one archery range and one stables. In the case they are already built, the option is not shown to the user.
- upgradeBuilding
  This method is used to upgrade buildings in the village. It works similar to the buildBuilding method. The buildings available for upgrade are shown using the printNumberedBuildings method and the user can make their choice. If they have enough resources, then the level of the building is increased by 1 and the cost is deducted from the resources.
- trainTroops
  This method is used to train troops in the village. Each type of troop can only be trained if their building has been built (barracks for swordsmen, archery range for archers and stables for cavalry). If the building has not been built yet, then the user is not given the option to train that troop. The maximum amount of troops that can be trained at once is determined by the level of the building. If the user enters a valid number, the troop amount is increased and the cost to train is reduced from the resources.
- createArmy
  This method is used to create and send an army to attack. It starts off by asking the user how many of each troop type they want in their army. Each amount is then added to the army troops and decreased from the village troops. Next, the user is asked which player they would like to attack. The stats for the army are updated using the updateStats and setTargets methods and the army is added to the map using addArmy.
- resourceLoop
  This method is used to loop through each building and generate the resource for that round. The if condition makes sure that only those buildings which are an instance of ResourceBuilding have the method generateResource executed on them.
- armyArrival
  This method is used when an army arrives back to its home village. It takes any resources that were stolen and add them to the village's resources and takes any surviving troops and puts that back in the village's troops.

Resources

This class contains the Resources. Both the villages and armies make use of this class. There are get methods for each variables. Instead of a normal set method, there are increase and decrease methods where the method takes in the amount that needs to be added/subtracted. There are also the following methods:

- startingResources
  This method is used when each village is declared. It updates the food, wood, and metal values to where the user has enough of each to build each resource-generating building.

- printResources
  This method is used to display the resources to the user.

<u>TroopStats</u>

In this class, the three different types of troops and their stats are declared. There are a number of get methods and, similar to Resources, increase/decrease methods instead of set methods. Inherited from this class are also the Swordsmen, Archers, and Cavalry classes. These classes do not have any methods, but only have the stats for each troop in the constructor.

<u>Troops</u>

A similar approach to that of the Resources class was taken for the Troops class. The class has get methods for each troop and also increase/decrease methods to modify the amounts. Inherited from this class is the Army class.

<u>Army</u>

The army class contains all the variables required to perform an attack on the village. It contains multiple get and set methods to be able to manipulate these variables. There are also the following methods found in this class:

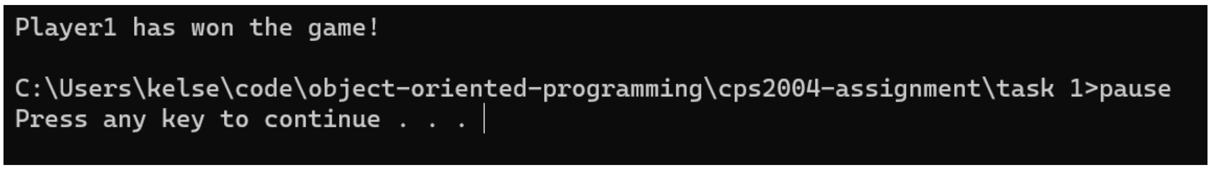
- updateStats
  This method is called anytime there is a decrease in the army troops. It updates the total attack, total health and carrying capacity according to the amount of troops. It also modifies the marching speed by setting it to the speed of the slowest troop.
- friendlyArrival
  This method is used when the army arrives back at the village. It calls the armyArrival method from the Village class.
- attackVillage
  This method is used to resolve an attack between an army and a village. The troops that are killed are decided at random. The method returns true when the attack of the army succeeds or a false when it fails. The Boolean is then used to determine the course of action in the traverseArmies method.

<u>User Guide</u>

The program is first compiled by opening the compile.bat file. This will create multiple Class files as required. The program can then be run by opening the run.bat file. It is important that this is done in that order as otherwise the program will not run. It is also recommended that the console window is maximised in order for the map and all the details to show fully without the user having to scroll.

The user can play the game by inputting their choices when prompted. The game is made up of multiple menus where the user can choose what they want to do in that round. There is also a 'Back' option on the menus so the user can go back a step if they decide to. This is helpful in case the user enters a wrong choice or changes their mind about performing a certain action.

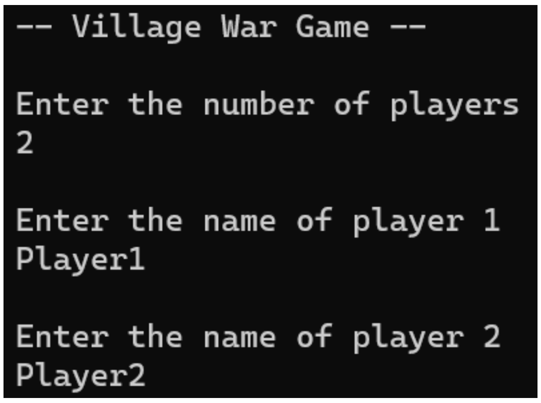
When the program finishes the pause command is executed in the command line. This is so the command window stays open, and the user has a chance to view the results of the game. Pressing any key will then close the window.



```
Player1 has won the game!

C:\Users\kelse\code\object-oriented-programming\cps2004-assignment\task 1>pause
Press any key to continue . . .
```

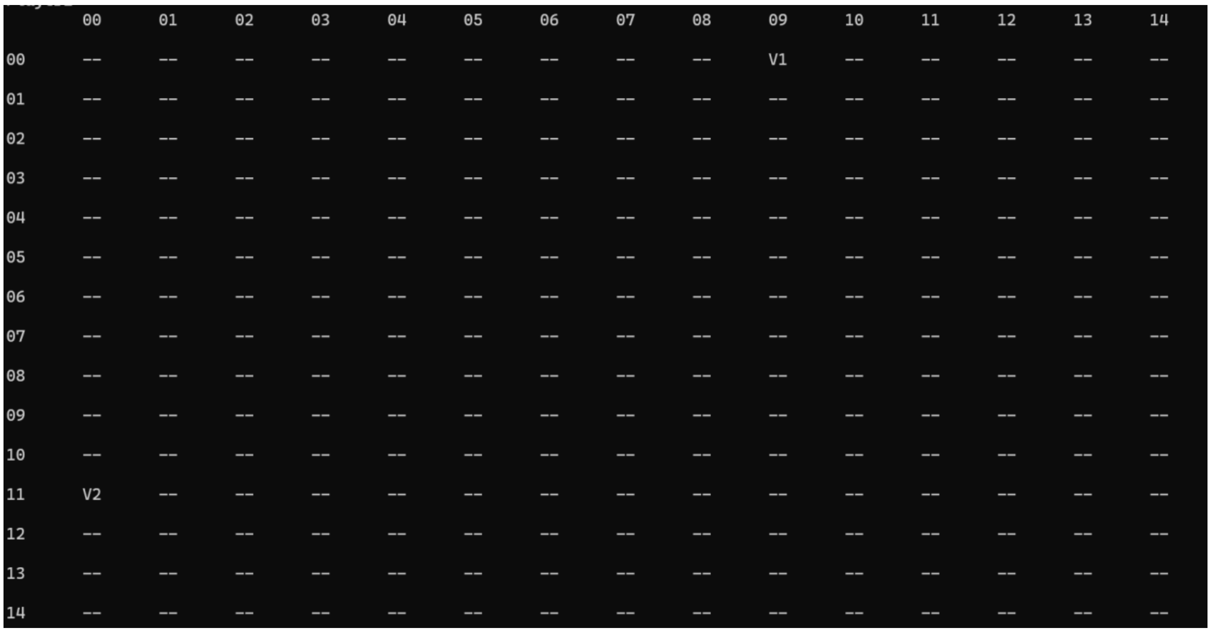
<u>Test Cases</u>

For the first test case, a village getting destroyed will be shown with two players: Player1 and Player2.



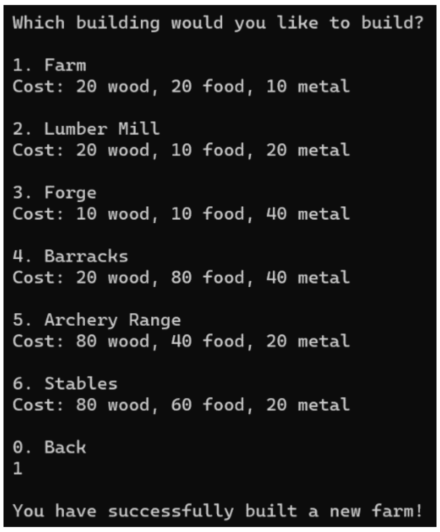
```
-- Village War Game --

Enter the number of players
2

Enter the name of player 1
Player1

Enter the name of player 2
Player2
```

The following is the map that was generated with the two villages on it.

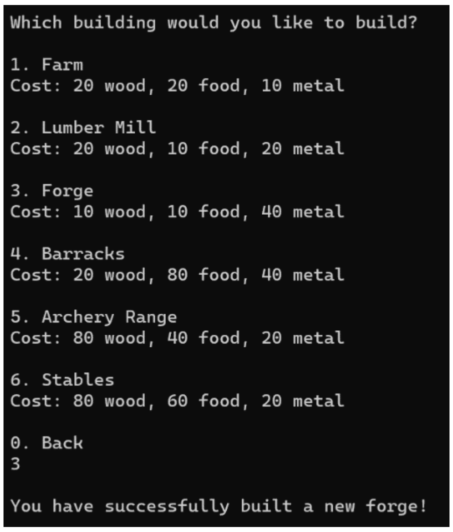


```
     00   01   02   03   04   05   06   07   08   09   10   11   12   13   14
00   --   --   --   --   --   --   --   --   --   V1   --   --   --   --   --
01   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
02   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
03   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
04   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
05   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
06   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
07   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
08   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
09   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
10   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
11   V2   --   --   --   --   --   --   --   --   --   --   --   --   --   --
12   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
13   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
14   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
```

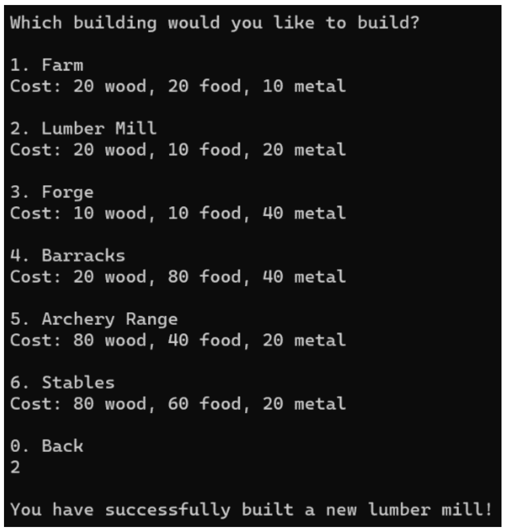
For Player1’s turn, the farm, the forge and the lumbermill were built.



```
Which building would you like to build?

1. Farm
Cost: 20 wood, 20 food, 10 metal

2. Lumber Mill
Cost: 20 wood, 10 food, 20 metal

3. Forge
Cost: 10 wood, 10 food, 40 metal

4. Barracks
Cost: 20 wood, 80 food, 40 metal

5. Archery Range
Cost: 80 wood, 40 food, 20 metal

6. Stables
Cost: 80 wood, 60 food, 20 metal

0. Back
1

You have successfully built a new farm!
```



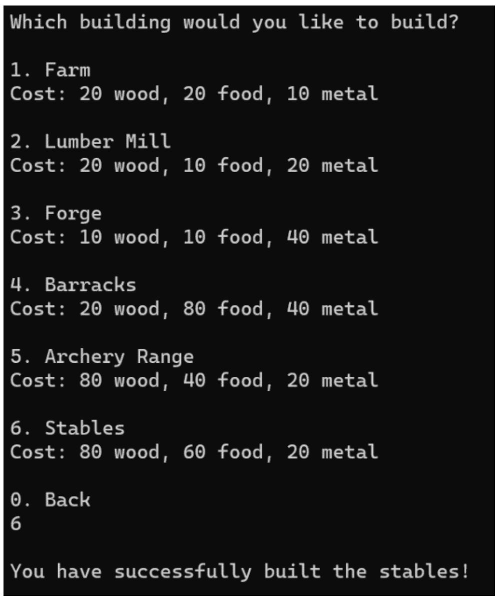
```
Which building would you like to build?

1. Farm
Cost: 20 wood, 20 food, 10 metal

2. Lumber Mill
Cost: 20 wood, 10 food, 20 metal

3. Forge
Cost: 10 wood, 10 food, 40 metal

4. Barracks
Cost: 20 wood, 80 food, 40 metal

5. Archery Range
Cost: 80 wood, 40 food, 20 metal

6. Stables
Cost: 80 wood, 60 food, 20 metal

0. Back
3

You have successfully built a new forge!
```



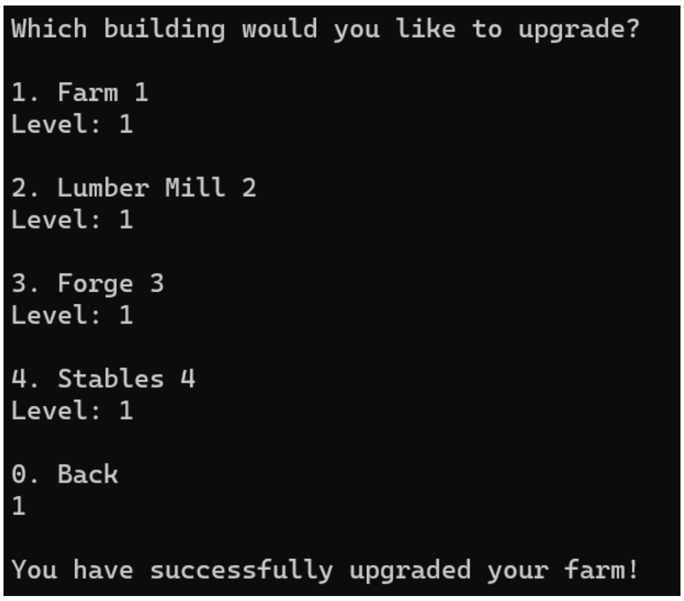
```
Which building would you like to build?

1. Farm
Cost: 20 wood, 20 food, 10 metal

2. Lumber Mill
Cost: 20 wood, 10 food, 20 metal

3. Forge
Cost: 10 wood, 10 food, 40 metal

4. Barracks
Cost: 20 wood, 80 food, 40 metal

5. Archery Range
Cost: 80 wood, 40 food, 20 metal

6. Stables
Cost: 80 wood, 60 food, 20 metal

0. Back
2

You have successfully built a new lumber mill!
```

Player1’s turn was then passed, and the same actions were taken for Player2. A few rounds were skipped to allow resources to accumulate.

In Player1’s turn, the stables were built and the farm was upgraded to level 2.

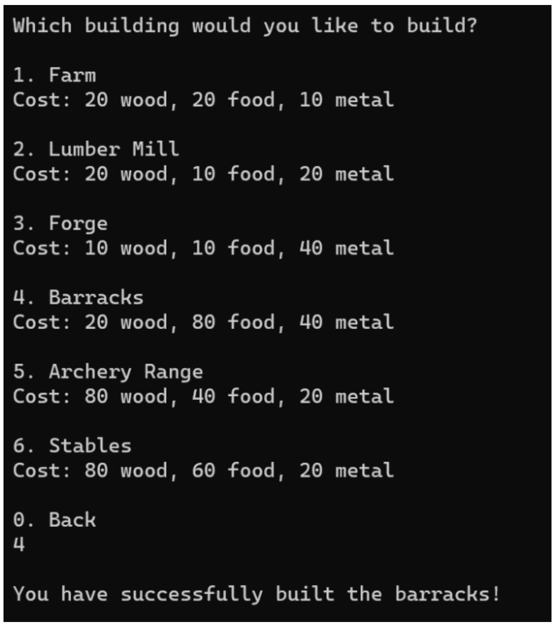


```
Which building would you like to build?

1. Farm
Cost: 20 wood, 20 food, 10 metal

2. Lumber Mill
Cost: 20 wood, 10 food, 20 metal

3. Forge
Cost: 10 wood, 10 food, 40 metal

4. Barracks
Cost: 20 wood, 80 food, 40 metal

5. Archery Range
Cost: 80 wood, 40 food, 20 metal

6. Stables
Cost: 80 wood, 60 food, 20 metal

0. Back
6

You have successfully built the stables!
```



```
Which building would you like to upgrade?

1. Farm 1
Level: 1

2. Lumber Mill 2
Level: 1

3. Forge 3
Level: 1

4. Stables 4
Level: 1

0. Back
1

You have successfully upgraded your farm!
```

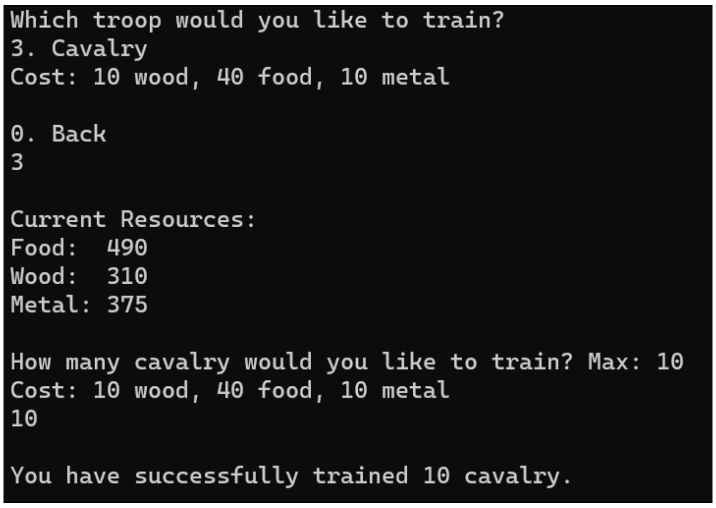
Player1's turn was then passed. In Player2's turn, the barracks were built.



```
Which building would you like to build?

1. Farm
Cost: 20 wood, 20 food, 10 metal

2. Lumber Mill
Cost: 20 wood, 10 food, 20 metal

3. Forge
Cost: 10 wood, 10 food, 40 metal

4. Barracks
Cost: 20 wood, 80 food, 40 metal

5. Archery Range
Cost: 80 wood, 40 food, 20 metal

6. Stables
Cost: 80 wood, 60 food, 20 metal

0. Back
4

You have successfully built the barracks!
```

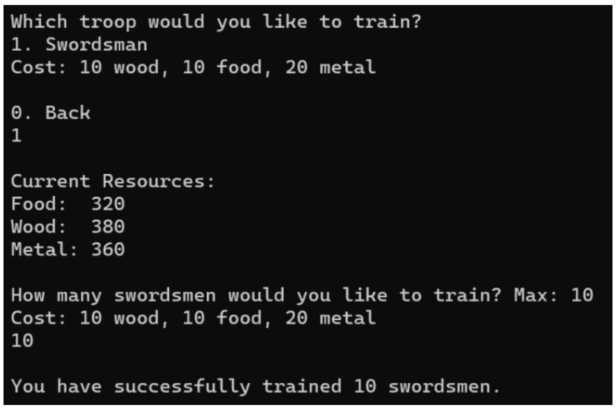
A few more rounds were passed to allow resources to accumulate.

In Player1's turn, 10 cavalry troops were trained and then the turn was passed.

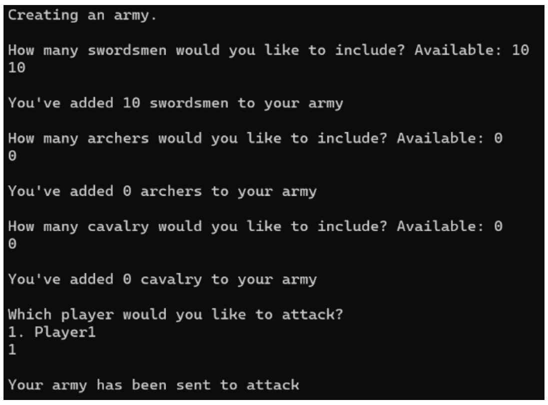


```
Which troop would you like to train?
3. Cavalry
Cost: 10 wood, 40 food, 10 metal

0. Back
3

Current Resources:
Food:  490
Wood:  310
Metal: 375

How many cavalry would you like to train? Max: 10
Cost: 10 wood, 40 food, 10 metal
10

You have successfully trained 10 cavalry.
```

In Player2's turn, 10 swordsmen troops were trained and sent to attack Player1's village. Player2's turn was passed.

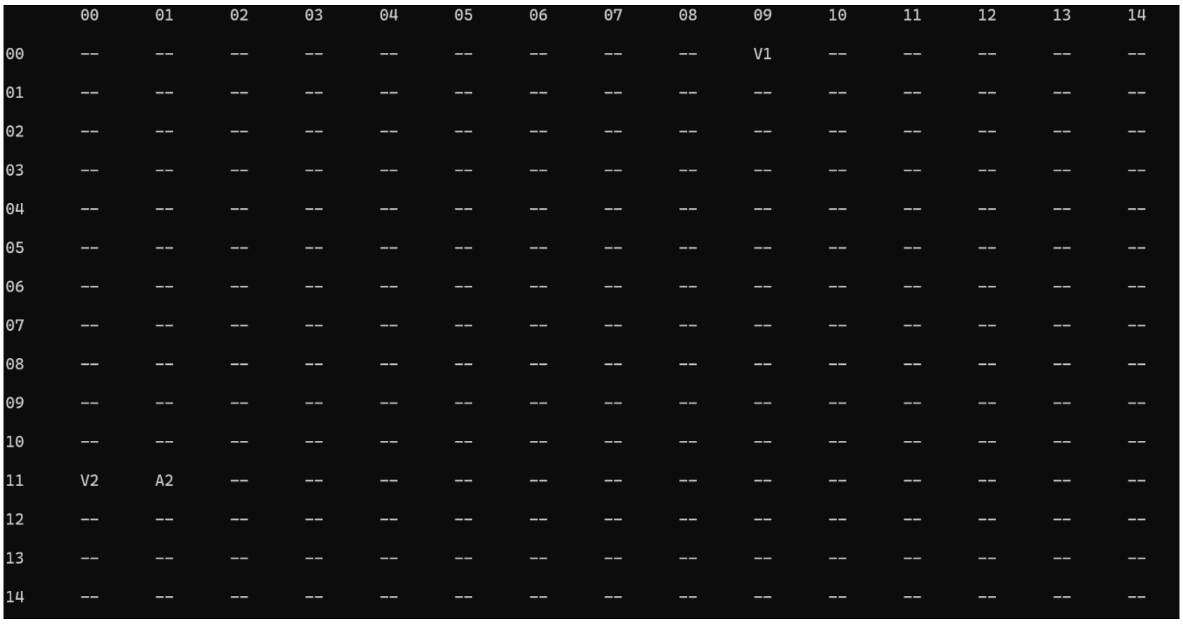


```
Which troop would you like to train?
1. Swordsman
Cost: 10 wood, 10 food, 20 metal

0. Back
1

Current Resources:
Food:  320
Wood:  380
Metal: 360

How many swordsmen would you like to train? Max: 10
Cost: 10 wood, 10 food, 20 metal
10

You have successfully trained 10 swordsmen.
```



```
Creating an army.

How many swordsmen would you like to include? Available: 10
10

You've added 10 swordsmen to your army

How many archers would you like to include? Available: 0
0

You've added 0 archers to your army

How many cavalry would you like to include? Available: 0
0

You've added 0 cavalry to your army

Which player would you like to attack?
1. Player1
1

Your army has been sent to attack
```

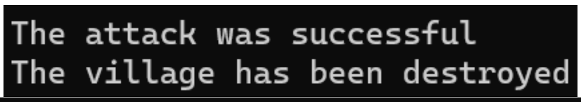
The army can be seen moving towards Player1’s village from here onwards.



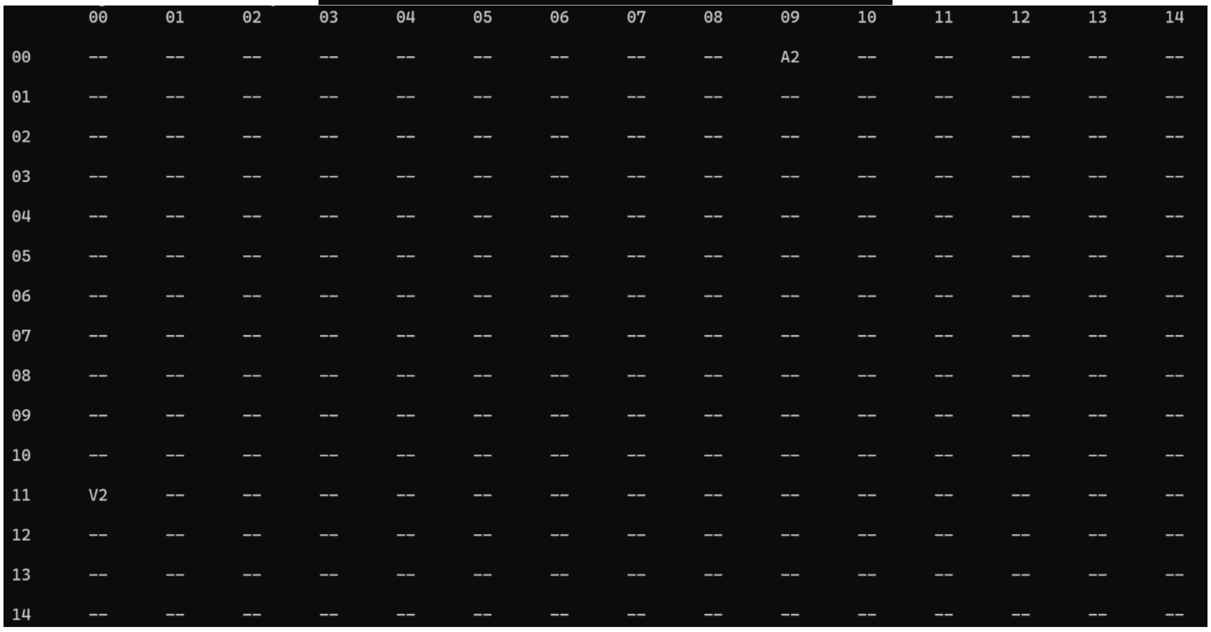
```
     00   01   02   03   04   05   06   07   08   09   10   11   12   13   14
00   --   --   --   --   --   --   --   --   --   V1   --   --   --   --   --
01   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
02   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
03   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
04   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
05   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
06   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
07   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
08   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
09   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
10   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
11   V2   A2   --   --   --   --   --   --   --   --   --   --   --   --   --
12   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
13   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
14   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
```

A few rounds were passed to allow the army to travel to the village.

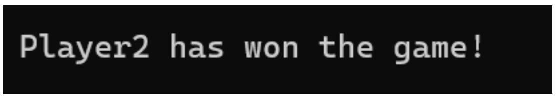
When the army reached the village, the attack was successful, and the village was destroyed. Resulting in Player2 winning the game. In the output of the map, V1 is now gone.



```
The attack was successful
The village has been destroyed
```



```
     00   01   02   03   04   05   06   07   08   09   10   11   12   13   14
00   --   --   --   --   --   --   --   --   --   A2   --   --   --   --   --
01   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
02   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
03   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
04   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
05   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
06   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
07   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
08   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
09   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
10   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
11   V2   --   --   --   --   --   --   --   --   --   --   --   --   --   --
12   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
13   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
14   --   --   --   --   --   --   --   --   --   --   --   --   --   --   --
```



```
Player2 has won the game!
```

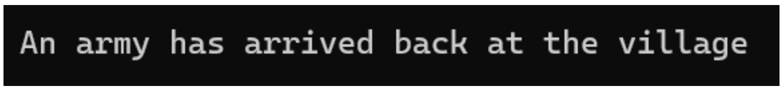
For the second test case, an army winning the battle but not destroying the village will be shown using Player1 and Player2.

As was done in the previous test case, the farm, the lumber mill and, the forge were built for both players and a few rounds were passed to allow resources to accumulate.

For Player1, the stables were built, and 30 cavalry troops were trained.

For Player2, the barracks were built, and 10 swordsmen were trained. An attack on Player1's village was sent with 10 swordsmen.

The army that was sent eventually arrived back at the village with stolen resources which were added to Player2's village.



```
An army has arrived back at the village
```

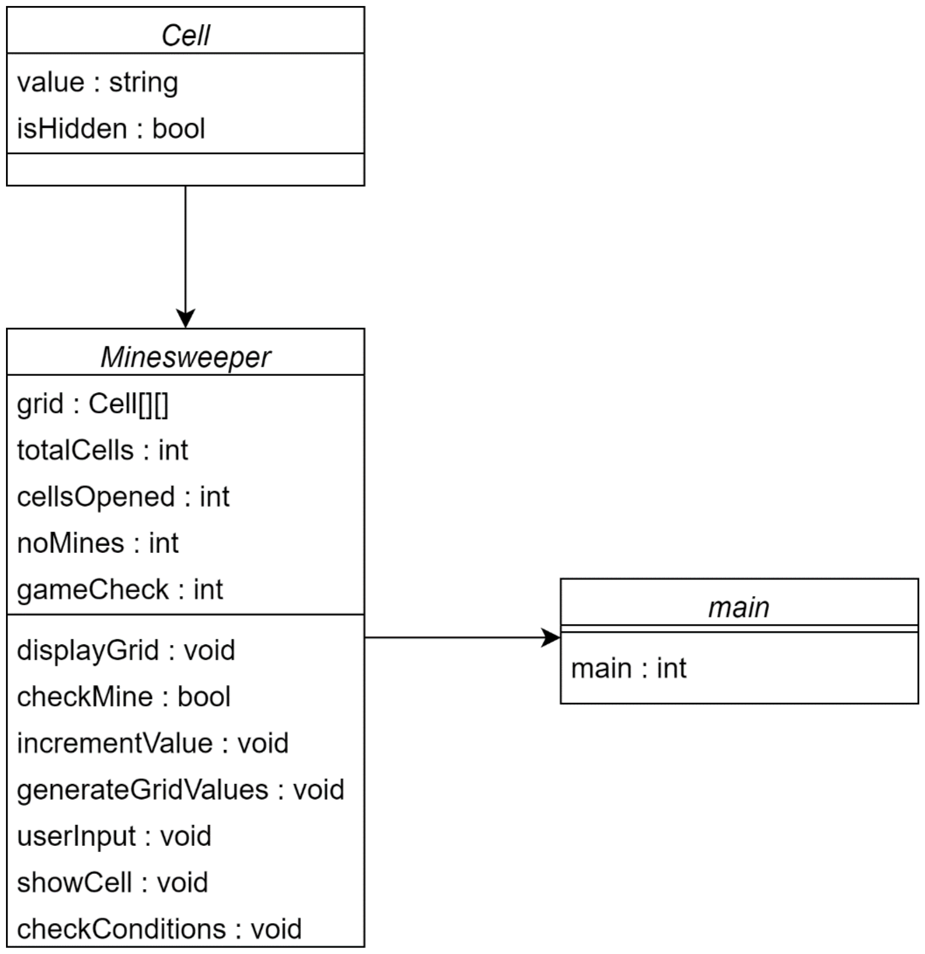
<u>Evaluation</u>

A major improvement that can be done to the game would be implementing a GUI. This would allow easier input from the user and any necessary data can be displayed at all times in more convenient places.

The AI was omitted due to the methods being heavily reliant on console input/output. Unfortunately, due to time constraints these methods could not be modified. This resulted in the implementation of AIPlayer and HumanPlayer to be redundant.

<u>Task 2 – Minesweeper</u>

<u>UML Diagram</u>



<u>Design and Implementation</u>

The Minesweeper was split into different classes as can be seen in the above diagram

<u>Cell</u>

This class contains the declaration for the cell that will be used in the grid. It does not contain any methods.

<u>Minesweeper</u>

This class contains all the methods which is required for the game to work.

- displayGrid
  This method is used to display the grid to the user in proper formatting.
- checkMine
  This method is used to check if a cell contains a mine. It returns true if it is a mine or false if it is not.
- incrementValue
  This method is used to increment the strings when required. This is done by changing the strings to an integer and then back to string.

- generateGridValues
  This method is used to generate the grid and randomly place bombs in it at the start of the game.
- userInput
  This method is used to accept in the user's input when required. It also checks for any invalid input or cells which have already been revealed.
- showCell
  This method is used to make the cell visible and also check if every cell that does not contain a bomb has been opened.
- checkConditions
  This method is used to check if the coordinates that were chosen correspond to a mine or not. If not and the cell happens to have no adjacent mines, then the 8 cells around it are revealed. If the cell has adjacent mines, then it is revealed showing how many mines are around it.

Main

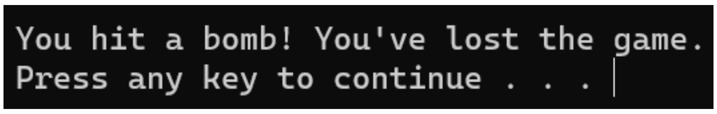
This class contains the main function which is run when the program is started. It calls the necessary method to initialise the grid and is responsible for the game loop. At the end of the main function in line 23 there is a command to stop the console from closing after the game finishes. This allows the user to the see the final results after either winning or losing the game.

User Guide

The program is first compiled by running the compile.bat file. This will a file called a.exe. The program can then be run by running the run.bat file. It is important that this is done in that order as otherwise the program will not run. It is also recommended that the console window is maximised in order for the grid to show fully without the user having to scroll.

The user can play the game by inputting the coordinates when prompted.

When the program finishes the pause command is executed in the main function. This is so the command window stays open, and the user has a chance to view the results of the game. Pressing any key will then close the window.

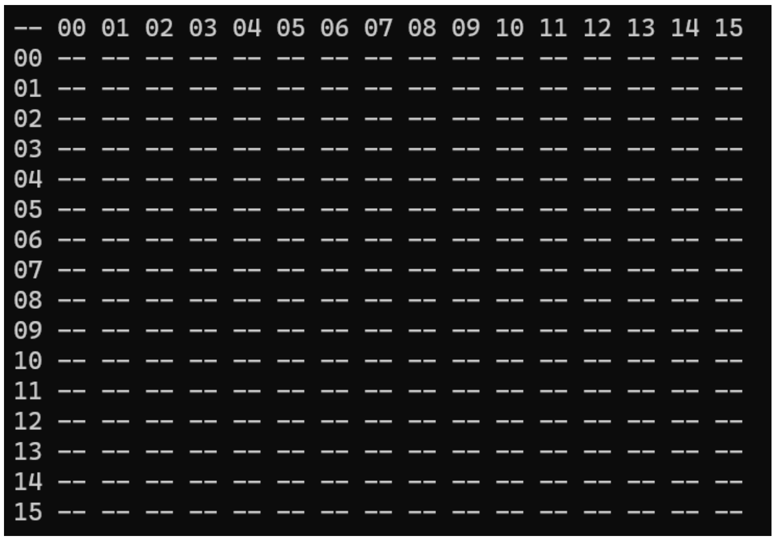


```
You hit a bomb! You've lost the game.
Press any key to continue . . .
```

Test Cases

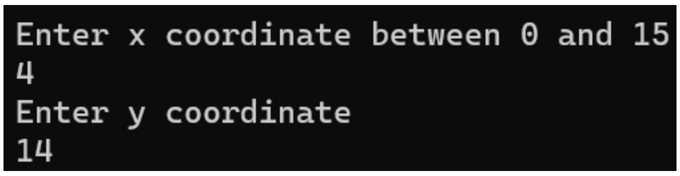
The first case that will be shown is when the user wins the game. For the sake of making the game easier, the number of mines was changed from 40 to 1 in the code of the game.

The game starts off by displaying the grid to the user.



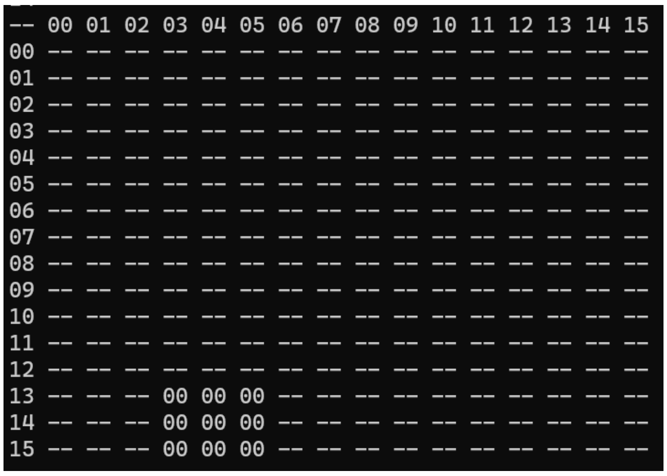
```
-- 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
01 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
02 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
03 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
04 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
05 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
06 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
07 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
08 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
09 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
11 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
12 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
13 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
14 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
15 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

The user is then prompted to enter x and y coordinates of their choice.



```
Enter x coordinate between 0 and 15
4
Enter y coordinate
14
```

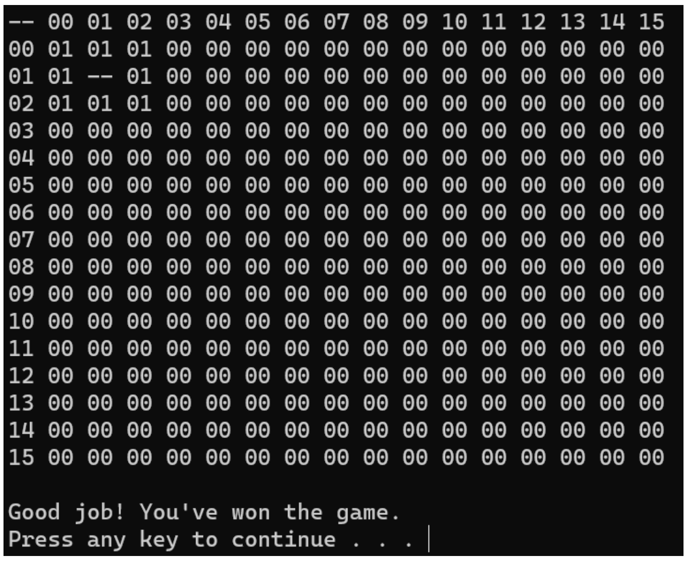
The coordinates (4, 14) were chosen, and that cell was revealed.



```
-- 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
01 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
02 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
03 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
04 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
05 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
06 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
07 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
08 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
09 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
11 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
12 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
13 -- -- -- 00 00 00 -- -- -- -- -- -- -- -- -- --
14 -- -- -- 00 00 00 -- -- -- -- -- -- -- -- -- --
15 -- -- -- 00 00 00 -- -- -- -- -- -- -- -- -- --
```

The user is then prompted again and keeps going until one of the conditions for the game to finish are met.

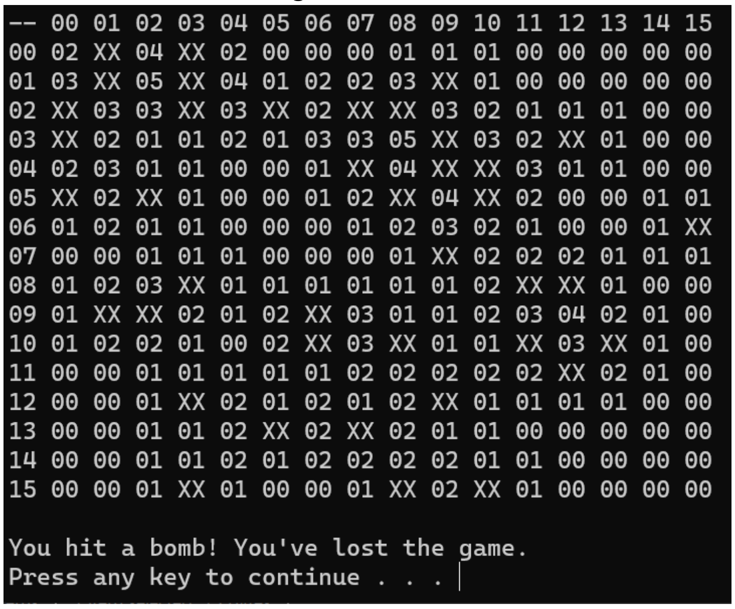
Here the bomb was not hit and therefore the game was won.



```
-- 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00 01 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00
01 01 -- 01 00 00 00 00 00 00 00 00 00 00 00 00 00
02 01 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00
03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
11 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
12 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
13 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
14 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
15 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Good job! You've won the game.
Press any key to continue . . .
```

For the second test, the game was played normally with 40 bombs in the grid and the game was lost.

In this case, after a bomb was hit the whole grid is revealed.



```
-- 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00 02 XX 04 XX 02 00 00 00 01 01 01 00 00 00 00 00
01 03 XX 05 XX 04 01 02 02 03 XX 01 00 00 00 00 00
02 XX 03 03 XX 03 XX 02 XX XX 03 02 01 01 01 00 00
03 XX 02 01 01 02 01 03 03 05 XX 03 02 XX 01 00 00
04 02 03 01 01 00 00 01 XX 04 XX XX 03 01 01 00 00
05 XX 02 XX 01 00 00 01 02 XX 04 XX 02 00 00 01 01
06 01 02 01 01 00 00 00 01 02 03 02 01 00 00 01 XX
07 00 00 01 01 01 00 00 00 01 XX 02 02 02 01 01 01
08 01 02 03 XX 01 01 01 01 01 01 02 XX XX 01 00 00
09 01 XX XX 02 01 02 XX 03 01 01 02 03 04 02 01 00
10 01 02 02 01 00 02 XX 03 XX 01 01 XX 03 XX 01 00
11 00 00 01 01 01 01 01 02 02 02 02 02 XX 02 01 00
12 00 00 01 XX 02 01 02 01 02 XX 01 01 01 01 00 00
13 00 00 01 01 02 XX 02 XX 02 01 01 00 00 00 00 00
14 00 00 01 01 02 01 02 02 02 02 01 01 00 00 00 00
15 00 00 01 XX 01 00 00 01 XX 02 XX 01 00 00 00 00

You hit a bomb! You've lost the game.
Press any key to continue . . .
```

Evaluation

A major improvement that can be done to the game would be implementing a GUI. This would allow easier input from the user by allowing the user to interact directly with the cells instead of entering their coordinates. This also reduces the chance of the user choosing the wrong mine.

Plagiarism Declaration Form

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I, the undersigned, declare that the assignment submitted is my work, except where acknowledged and referenced.

I understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected and will be given zero marks.

Kelsey Bonnici	K Bonnici
Student Name	Signature
CPS2004	Object Oriented Programming Assignment
Course Code	Title of work submitted
08/02/2023	
Date	