CS1006 W04 Practical 160004048, 160005600 Tutor: A. Toniolo 2017-02-10

# Wumpus

## Requirements

We were asked to create a ‘wraparound’ cave system. Then an adventurer, a wumpus (can be smelled from an adjacent location), an exit and a treasure(glitteringness appear on an adjacent location) have to be created in the content of the cave. Furthermore, some number of pits (can be felt by breeze in the adjacent location) and bats (transport to a random place) have to be added. The location of the content has to appear randomly and change every game. The adventurer has to collect treasure and reach an exit. In addition, the player dies if s/he enters a pit or meets the wumpus.

## System Overview

The basic requirements are done. In addition to that, arrows can be collected and the wumpus can be killed by a player. Furthermore, we have created a graphics based design with suitable pictures of all the positions. Next, an AI is created which always wins a game in an efficient way. Furthermore, when a treasure is coughed, a player can ask for a hint which makes an ‘Exit’ appear. The start page is created with two buttons (‘START AI DEMO’ and ‘PLAY GAME’) and a number of moves with a number of arrows are being counted and showed below. Furthermore, suitable messages are printed in the screen when losing, winning or catching the treasure.

## Design

The program creates a game which can be played by a user or by computer. There are objects created of each character (wumpus, bat, smell, breeze, treasure, pitfall, player, AI, tile). The area of the game is a 2D array where all the objects appear and a player moves. The game is graphics based so not only the values of a grid has to be updated, but also the images of a player and other objects depending on the values of a grid. Every object has it’s own picture(s) which change states depending on how a player moves. A player and an AI classes are closely related because these objects act similarly. The difference is that an AI moves automatically to the tiles which were explored less than the other ones. When a player is controlled by a user, the right commands are entered from a keyboard. Furthermore, the right messages are printed on the screen when the goal is achieved or a player dies. In addition, the number of moves and arrows, a message of a hint are printed below.

## Implementation

There are 11 classes created:

**Game class:**
There is a ‘main’ method where a frame with a screen is created. The name of the frame is ‘Hunt The Wumpus’.

**Treasure class:**
There is a constructor which creates a treasure given the two coordinates in the playing area. The coordinates are chosen randomly in a range from 4 to 15 so that the treasure will not appear near the border (because the player appears randomly on the border and it would be too easy to win the game if they appear very close to each other). Also integer values are given to all the 4 directions so that the glitteringness appear in the right place. All the variables are private thus getters and setters are used.

**Wumpus class:**
The class is almost identical to the ‘Treasure’ class because it the wumpus also appears somewhere in a range from 4 to 15.

**Pitfall class:**
There is a constructor which creates a pitfall given the two randomly chosen coordinates. As we have an array of P pitfalls, the constructor is called P times (P is chosen when creating pitfalls). Then, the 4 sides are equal to the remainder dividing by 20 (the dividing is necessary if the random coordinates are on the border).

**Arrow class:**
An Arrow constructor is created which takes two coordinates. Getters to reach the coordinates outside the class are created.

**Bat class:**
The class is almost identical to an ‘Arrow’ class. The constructor which creates a bat is called B times because we have an array of B bats. (B is chosen when creating bats).

**GridTiles class:**
2D grid is created to represent the area of the screen. Each element of the grid represents 30x30 pixels square. We have treasure, wumpus, pitfalls, arrow and bats created and put in the grid in the right locations. The grid initially is filled in with zeros. The method ‘setGridTile’ takes the coordinates of all the objects and the number which represents an object. The method changes the values of the 2D grid according to the randomly chosen locations of each object (breeze, smell and glitteringness inclusive).
When choosing coordinates of the objects, it is checked that the right squares are not taken by the other objects. Thus, there is ‘checkCoordinate’ method to check if the chosen 1 location is empty (when adding bats and arrows).
Next, there is a method ‘checkCoordinates’ which checks if all the 5 (the main point plus 4 sides) locations are empty (when adding a treasure, wumpus and pitfalls). Furthermore, there is a method ‘randomCoordinate’ which returns a value in a range 4-15 (inclusive) for giving coordinates to treasure and wumpus.
For the location of other objects, there is also a method ‘randomCoordinates’ for a random coordinate in the whole grid.
It is also worth mentioning that all the values of the given location in the grid which are checked in the ‘setGridTile’ method have already been changed before so there is no need to change them again.
Finally, there is an ‘exitAppears’ method which chooses an empty place (no matter if it was or was not explored before) for Exit to be.

**Tile class:**
The Tile constructor takes an image and a boolean vales which shoves the state of the tile. If the player has explored the location before, the value of ‘blocked’ variable is ‘true’ and the right picture appears. Else, the tile is a black image.

**Player class:**
An object of the GridTiles class is created so that a player would have a space to move. There is a constructor which takes the GridTiles object and adds a player to the grid and gives a speed to the player.
There is also a method ‘start position player’ which sets random coordinates (near the border) to the player. The ‘do while’ loop is repeated until an empty grid tile is found.
Furthermore, there are methods which return the right boolean values if a player is dead, a player has won a game, an arrow is found and a treasure is found. These values are used for printing the right message on the screen.
Next, there is an ‘update’ method which is called after each move is made. A value of the grid tile on which the player stands is changed by the other value so that the right image would appear and the right functionality would be done. Also, inside the method, the ‘do while’ loop is used to look for a suitable random location when a bat is found and a player has to be moved to a random place. When a treasure is found, the boolean value of the ‘playerWon’ variable is given so that the exit will appear since that moment. In addition, the ‘false’ values are given to all the directions so that the player would look for another direction when making the next move.
In the ‘draw’ method, a right image is drawn on the player’s tile.

**Screen class:**
Screen class is a subclass of Jpanel which implements Runnable and KeyListener functions.
Inside the class, the values of the screen size are given. In addition, the suitable boolean values are given to the ‘gameStart’, ‘computer’, ‘menuPick’ and ‘startSetUp’ so that the right screen would be printed. The boolean value of computer shows if a user plays the game or an AI. An AI is created in the player class because it uses the same methods as a player.
Furthermore, the ‘addNotify’, ‘keyTyped’, ‘keyReleased’ and ‘keyPressed’ are necessary for the functionality of the game. The thread is created and the key entered from a keyboard is read in the ‘addNotify’ method. Next, the ‘keyPressed’ method takes the KeyEvent object and gives a new direction to the player, starts the game, changes menu picture, shoots to the right direction or shows a hint to the player (according to the key entered). The ‘keyReleased’ method is similar to the ‘keyPressed’ one. Both of them are important for the player to stop after each move until the next key is pressed.
Moreover, the method called ‘getMovesMade’ is used to count the steps made. The number of moves is updated after each move is made.
There is a ‘paint’ method which writes the messages below. It uses the methods in the player class and prints the number of moves made and the amount of arrows. The font and colours of strings are chosen inside the method.
In addition, the ‘render’ method is created to print suitable messages on the screen.
The ‘render’ method is called to print the right start screen with the right images and sizes. It is called after each step so that the ‘draw’ methods of the objects of GridTiles and Player classes would be called (to fill in the grid) together with a paint method (to write the correct messages).
Finally, one of the most important methods of the game is the ‘run’ method which has a while method which calls the ‘update’, ‘render’ and ‘draw’ methods while a game is running.

**AI class:**
In the class, the methods of an AI are created which help him/her to move automatically to the right directions and always win.
An AI is the same player which moves automatically. The idea of an AI move decisions is that he/she moves to the adjacent grid which was explored the least times. To do that, 2D array of gridMoves is created (the same size as the gridTiles array) which elements represent how many times an AI has visited each grid.
There is a ‘makeMove’ method which takes a Player object and an object of the GridTiles class. Inside the method, it is checked if the grid tile player stands is not a breeze and not a smell.
In that case the player goes to the side which represents the smallest number of visits.
In case it is a breeze or a smell, the player moves one step back and chooses one of the 2 latest used directions (chooses right or left if the step back was up or down, chooses up or down if the step back was right or left). Moreover, the number of moves to the one side is chosen randomly from 1 to 4.
In case the player stands on the glitteringness tile, he/she goes one step back and starts exploring all the four adjacent tiles until the treasure is found.
Next, when a treasure is found, all the gridMoves elements are started counting again from zero. It makes the player more efficient when looking for an ‘Exit’.

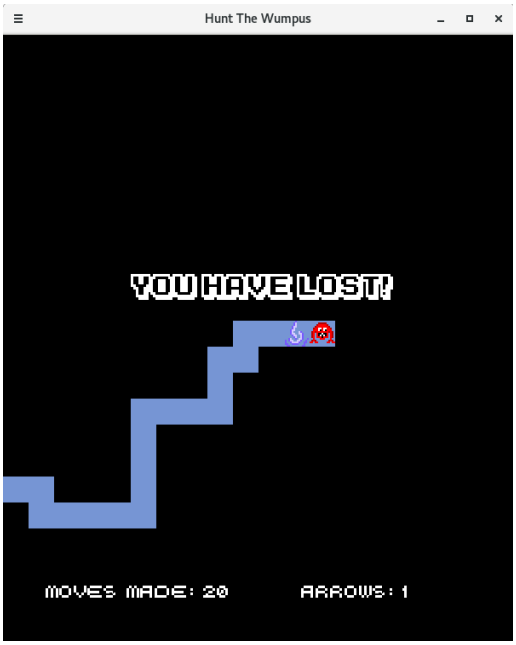
# Testing

When the start page appears there are two buttons which can be chosen. The grid appears when a space key is pressed. (1)

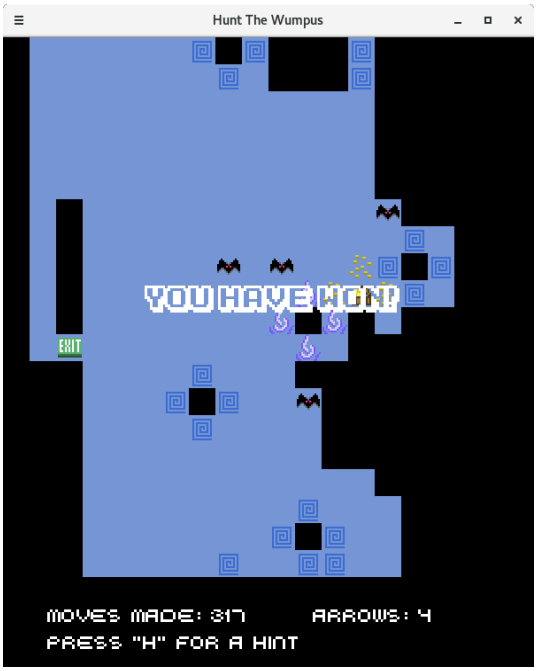
A player can be moved using down, up, left, right keys. A player does not move further if a key is pressed for a longer time. A breeze, smell, treasure, glitteringness, bat and a player appears when a tile is visited. Number of moves and number of arrows are counted and printed below. (2)


A suitable message is printed when a treasure is found.

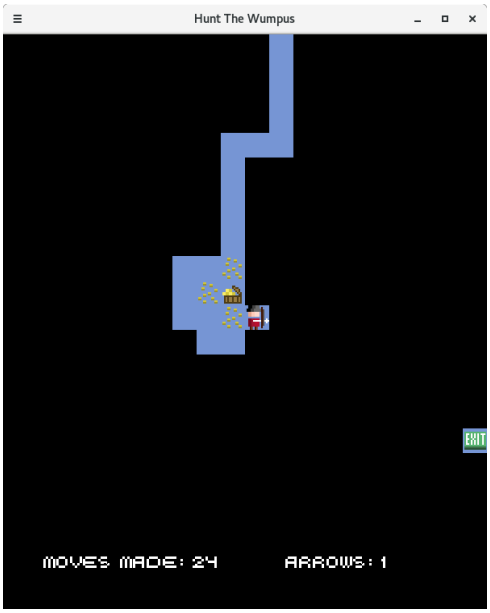

A suitable message is printed when a player loses. (4)



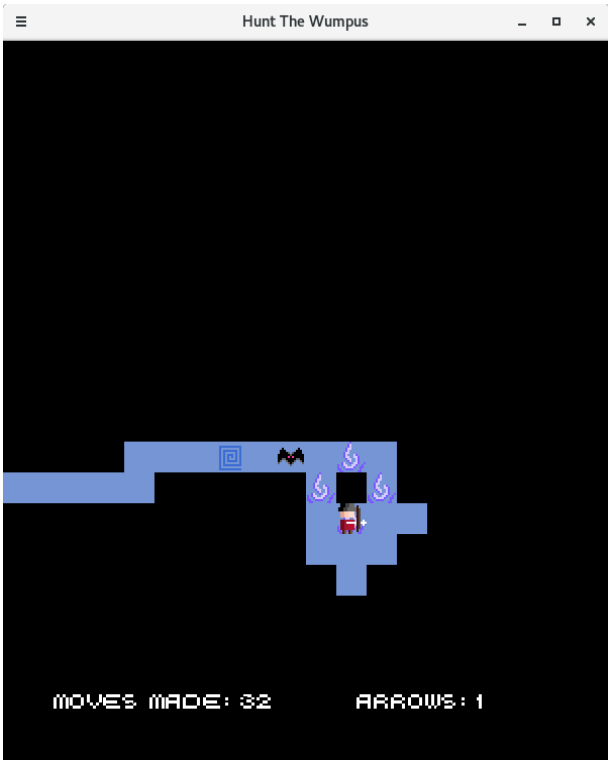
A suitable message is printed when a player wins. Also, it is seen that a ‘hint’ option appears below. (5)

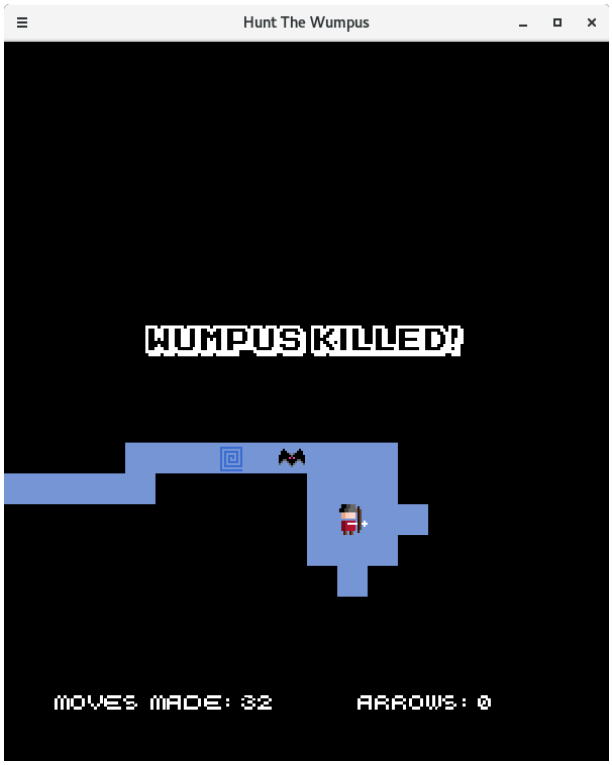



An exit appears when ‘h’ or ‘H’ is pressed.

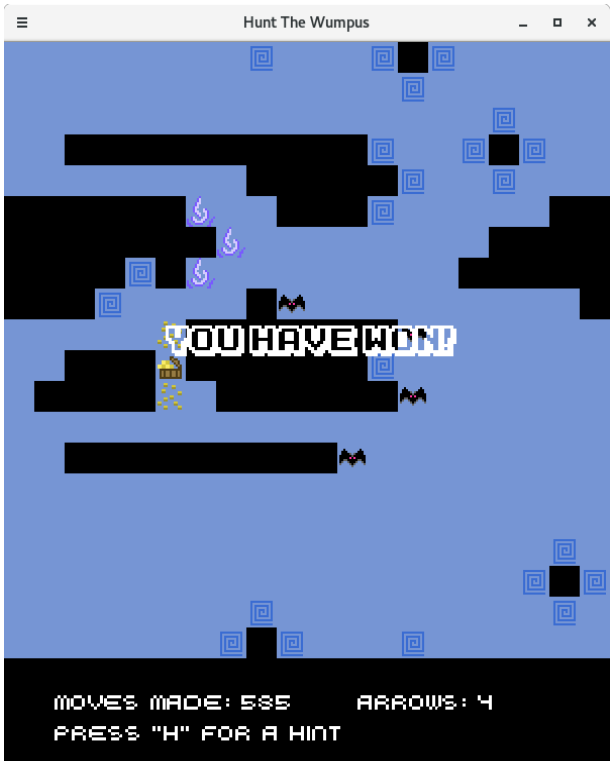


When wumpus is found and a right key is pressed, the wumpus dies. (‘W’ was pressed in the example). Also, a number of arrows is decreased by 1 when shoot.




An AI demo works and always wins the game. It is also seen that the tile with 3 arrows was coughed because the number of arrows is increased by 3.



**Evaluation**

The basic requirements are satisfied. In addition, the extensions are done: the game is graphics based, an AI is created, the wumpus can be shoot and an arrows are coughed. Furthermore, a hint can be asked. Suitable messages are printed during the process of the game. All in all, the practical is fully done. Although, some parts can still be improved.

**Individual contribution**

We have worked together in labs almost all the time. The basic code was made by Mantas.

Vaiva Augustinaite: Implemented AI structure.  I was responsible for creating AI exploration algorithms and developing final contributions to computer controlled player.

Mantas Skackauskas: game code, graphics, movement, design structure.