TP3 Project Updates (Final)

### Project description

QuackQuest: Midnight Reunion

This project involves the game user controlling a mother duck navigating through a maze in the darkness with limited visibility to find her baby ducks. As time passes, more of the maze becomes visible until the entire maze is revealed, signalling the end of the game. If it is daytime and you haven’t collected all the baby ducks, you will lose the game. Moreover, the mother duck can break through the walls (limited though), so it is your chance to find the shortest path to finish the task before the time ends. There are two game modes, generated respectively by DFS and Prim's. The second mode is relatively harder because there are more dead ends. There are also 2 player versions within each mode. Have fun saving the ducks!

### Similar projects

* Pac-Man is an iconic maze game where users control the movement of the character through a maze and evade enemies along the way. This game is similar since it also involves maze navigation and collecting items. However, QuackQuest involves extra features such as limited visibility and teleportation. Moreover, this game ends after the mother duck and all her baby ducks exit the maze, whereas Pac-Man ends after eating all the dots.
* Dead by Daylight features a killer and countdown timer in which the game automatically ends if it is daytime. The players are also required to repair generators to exit. Rather than repairing generators in Dead by Daylight, QuackQuest involves collecting baby ducks along the way.

### Structural Plan

* Main Python file: main.py
  + Combines all files
  + Draws the lobby and other features that enhance the UI experience, including buttons and animations
* Maze printing file: printMaze.py
  + Prints out the maze and randomly generates baby ducks
* General features file: duckClasses.py
  + Storing the methods I wrote using OOP
* Maze generation file: generateMaze2.py using DFS
  + All code required to randomly generate the maze using DFS
* Maze generation file: generateMaze1.py using Prim’s
  + All code required to randomly generate the maze using Prim’s

### Algorithmic Plan

The trickiest part of the project is the initial random generation of the maze. Recursive algorithms need to be implemented to ensure that the generated maze has a viable solution while it makes random choices at each step. In this project, there are various algorithms that I am going to use to generate different game modes. The first one uses depth-first search and starts at a random cell in the board, marks it as visited and acquires information about its neighbours. If the neighbour cell has not been visited, I will remove the wall between this cell and the neighbour and set the neighbour as the current cell, which allows me to make another recursive call. If a dead end is encountered, the algorithm will start backtracking.

Prim’s algorithm was trickier to understand. I was not familiar with minimum spanning trees and how to ‘grow’ the maze by connecting neighbouring cells with their random neighbours by selecting unvisited walls. Moreover, minimum spanning trees require choosing the edge with the minimum weight and I struggled to come up with an approach to assign weights to each edge. Therefore, I ended up modifying the algorithm into a slightly simpler form where every edge is considered equal. At every step, the algorithm chooses an arbitrary edge.

### Timeline Plan

|  |  |
| --- | --- |
| Date | Progress |
| 11/26 | Finish generating mazes 1 |
| 11/27 | Finish randomly generating the ducks, visibility circle |
| 11/30 | Finish lobby, instructions page, special power, duck animation |
| 12/01 | Finish 2-player mode |
| 12/03 | Finish generating maze 2 using Prim’s algorithm |
| 12/05 | Readme File, video demo, finalise design document for TP3 submission |

### Version Control Plan

I back up my code using a GitHub repository.

A screenshot of a computer

Description automatically generated

### Module List

This project does not involve any external modules/hardware/technologies.

TP2 Project Updates

### Project description

QuackQuest: Midnight Reunion

This project involves the game user controlling a mother duck navigating through a maze in the darkness with limited visibility to find her baby ducks and exit the maze. There will be two stages: 1) finding the baby ducks and 2) returning to their house (exiting the maze). As time passes, more of the maze becomes visible until the entire maze is revealed, signalling the end of the game. If it is daytime and you haven’t collected all the baby ducks, you will lose the game. Moreover, the mother duck can break through the walls (limited though), so it is your chance to find the shortest path to finish the task before the time ends. Have fun saving the ducks!

### Similar projects

* Pac-Man is an iconic maze game where users control the movement of the character through a maze and evade enemies along the way. This game is similar since it also involves maze navigation and collecting items. However, QuackQuest involves extra features such as limited visibility and teleportation. Moreover, this game ends after the mother duck and all her baby ducks exit the maze, whereas Pac-Man ends after eating all the dots.
* Dead by Daylight features a killer and countdown timer in which the game automatically ends if it is daytime. The players are also required to repair generators to exit. Rather than repairing generators in Dead by Daylight, QuackQuest involves collecting baby ducks along the way.

### Structural Plan

* Main Python file
  + Combines all files
* Maze printing file: printMaze.py
  + Prints out the maze and randomly generates baby ducks
* General features file: drawPage.py
  + Draws the lobby and other features that enhance the UI experience, including buttons and animations
* Maze generation file: generateMaze2.py using DFS
  + All code required to randomly generate the maze
  + Code related to how the wolf tracks the duck
* Maze generation file: generateMaze1.py using Prim’s

### Algorithmic Plan

The trickiest part of the project is the initial random generation of the maze. Recursive algorithms need to be implemented to ensure that the generated maze has a viable solution while it makes random choices at each step. In this project, there are various algorithms that I am going to use to generate different game modes. The first one uses depth-first search and starts at a random cell in the board, marks it as visited and acquires information about its neighbours. If the neighbour cell has not been visited, I will remove the wall between this cell and the neighbour and set the neighbour as the current cell, which allows me to make another recursive call. Moreover, there will be another two game modes, one using Prim’s algorithm, creating more dead ends instead of long passages.

### Timeline Plan

|  |  |
| --- | --- |
| Date | Progress |
| 11/26 | Finish generating mazes 1 |
| 11/27 | Finish randomly generating the ducks, visibility circle |
| 11/30 | Finish lobby, instructions page, special power, duck animation |
| 12/01 | Finish 2-player mode |
| 12/03 | Finish generating maze 2 using Prim’s algorithm |
| 12/06 | Readme File, video demo, finalise design document for TP3 submission |

### Version Control Plan

I back up my code using a GitHub repository.

A screenshot of a computer

Description automatically generated

### Module List

This project does not involve any external modules/hardware/technologies.

TP0 Project Proposal

### Project description

QuackQuest: Midnight Reunion

This project involves the game user controlling a mother duck navigating through a maze in the darkness with limited visibility to find her baby ducks and exit the maze. There will be two stages: 1) finding the baby ducks and 2) returning to their house. As time passes, more of the maze becomes visible until the entire maze is revealed, signalling the end of the game. Overall, there are several ways that you can lose: mother duck being caught by the wolf, it is daylight, or you left another duck in the maze after exiting. There will always be a duck at the bottom right and after you collect that duck, the wolf will be triggered and will hunt down the duck at a faster speed after 5 seconds. Moreover, the mother duck can break through the walls (limited though). Have fun saving the ducks!

### Similar projects

* Pac-Man is an iconic maze game where users control the movement of the character through a maze and evade enemies along the way. This game is similar since it also involves maze navigation and collecting items. However, QuackQuest involves extra features such as limited visibility and teleportation. Moreover, this game ends after the mother duck and all her baby ducks exit the maze, whereas Pac-Man ends after eating all the dots.
* Dead by Daylight features a killer and countdown timer in which the game automatically ends if it is daytime. The players are also required to repair generators to exit. Rather than repairing generators in Dead by Daylight, QuackQuest involves collecting baby ducks along the way.

### Structural Plan

* Main Python file
  + Introduction screen with title, game instructions, and buttons for difficulty selection to increase user experience
  + Engaging small animations such as stars
* Maze generation file
  + All code required to randomly generate the maze
  + Code related to how the wolf tracks the duck

### Algorithmic Plan

The trickiest part of the project is the initial random generation of the maze. Recursive algorithms need to be implemented to ensure that the generated maze has a viable solution while it makes random choices at each step. In this project, there are various algorithms that I am going to use to generate different game modes. The first one uses depth-first search and starts at a random cell in the board, marks it as visited and acquires information about its neighbours. If the neighbour cell has not been visited, I will remove the wall between this cell and the neighbour and set the neighbour as the current cell, which allows me to make another recursive call. Moreover, there will be another two game modes, one using Prim’s algorithm, creating more dead ends instead of long passages. Another tricky part of this project would be implementing the wolf in the single-player mode, which is an AI player that follows the ducks and is not allowed to cross any walls.

### Timeline Plan

|  |  |
| --- | --- |
| Date | Progress |
| 11/26 | Finish generating mazes 1 |
| 11/27 | Finish lobby, randomly generated the ducks |
| 11/30 | Finish the wolf’s algorithm and its movement before TP2 |
| 12/03 | Finish adding more special ability feature of breaking through the walls |
| 12/06 | Readme File, video demo, finalise design document for TP3 submission |

### Version Control Plan

I back up my code using a GitHub repository.

A screenshot of a computer

Description automatically generated

### Module List

This project does not involve any external modules/hardware/technologies.