

Introduction

The maze created is a culmination of different ideas from other games. The game is based on a rogue-like with permadeath the main objective being to survive by collect stars hidden within a randomly generated maze, fighting monsters and gaining levels. C++ was chosen for its Object orientated programming ability to create the tiles that make-up the maze and to have multiple entities occupying the maze at once.

Project design sheet

Review of existing programs:

Pac-Man:



Pac-man is an arcade game developed by Namco and released in 1980. Pac-man's successes can be attributed to its unique design compared to other games of its time. Pac-Man is also a demonstration of an abstract maze. The maze consists of 'walls' and 'corridors' which is very similar to a traditional maze however contains loops for added complexity which also means more manoeuvrability to avoid monsters. The map is also symmetrical down the middle. The objective of the game is to navigate Pac-man around the predetermined maze collecting Pac-dots and avoiding monsters to stay alive. Once all the Pac-dots are collected the level is completed and then the player is moved onto the next level. The Pac-man game can be solved many different ways depending on how you

collect the Pac-dots allowing for many strategies. One interesting feature the game contains is 'Pac-Pellets' which when consumed make the monsters vulnerable and allow Pac-man to 'eat' monsters and gain points for doing so. This adds a level of depth and strategy as the player needs to use these pellets when appropriate. However the pellets also provide the only scaling difficulty to each individual level. As the levels increase the amount of time a pellet will make monsters vulnerable is decreased. While this feature is reasonable it highlights that there is not much of a difficulty curve in the game. Each monster also has its own personality programmed meaning they react differently to one another. This keeps the enemies from becoming stale and predictable. Pac-man was also designed to be played infinitely to allow for limitless high scores and playing time. However due to a bug this is prevented from happening. All these features vary the gameplay and makes each play through of the game enjoyable which is evidenced by how popular the game is.

Image source: <https://en.wikipedia.org/wiki/Pac-Man>

Labyrinth (IOS/Android):



Labyrinth is game developed by Illusion Labs and played on IOS and android. Labyrinth uses the iPhone's tilt recognition software to manoeuvre a ball around an increasingly difficult set of mazes with 'holes' the ball can fall down resetting the level. The level is complete when the ball is placed into the finish hole. The game is based off the tradition Labyrinth board game released in 1946 which was the physical equivalent of the IOS and Android version. Labyrinth has a large number of levels to play with a large range of difficulty for players of all abilities. This means the game can be played by anyone. Another good feature about Labyrinth is its quick play time. It can be played while waiting for a bus or just a time filler when bored. Its difficulty curve challenges good players and encourages them to keep playing however gameplay is also very repetitive and similar which players may grow tired of. Labyrinth 2, the successor to Labyrinth fixes this problem by adding many new features including such as bumpers, cannons, slingshots, duplicators and many more add much more verity to the gameplay.

Image source: <http://www.apppicker.com/appsale/26312/top-ipad-games-gone-free-for-october-30th>

Dungeon Master (Atari ST):



Dungeon Master was developed by FTL Games and released on the Atari ST in 1987. Dungeon Master is a role-playing video game, with real time combat and turn based movement. The objective of the game is to explore the maze, solve puzzles and ultimately reach the end alive with your team through a series of levels. The game popularised many features that modern RPG's use today such as levelling a skill through use of that skill. Dungeon Master has a significant amount of content with unique monsters

Your characters which prevent the game from getting stale. However Dungeon master technically has one storyline which once played through is the same, this may become boring for some players. One of Dungeon Master's strongest features is its interactive puzzles which allows players to find clues or levers in the environment to solve the puzzles for rewards making players feel very involved in the game. Another example of this is the spell casting feature which requires users to remember or experiment with a sequences of runes to cast a spell. Dungeon Masters maze is relatively simple however with numerous puzzles to solve dispersed with a lot of combat it makes the game very enjoyable.

Image source: <http://www.myabandonware.com/game/dungeon-master-n0>

World's Hardest Game (PC):



World's Hardest Game is an online flash based video game considered to be very challenging. The objective is to complete a series of levels which you manoeuvre around dodging moving obstacles and collecting keys. The levels get increasingly harder throughout providing a good challenge for capable players. A negative point is that there is only 30 levels however this is solved by the developers creating subsequent iterations of the game with World's hardest game 2, 3 and 4. One of the games strongest points is its use of moving objects as

obstacles. It's very different to other maze games as it focuses on the player's quick reaction time and an ability to solve the maze with confusing moving objects. World's hardest game is very enjoyable and rewarding if you're willing to persevere through numerous deaths.

Image source: <http://www.onlinegamesector.com/onlinegame/18562/worlds-hardest-game-4.htm>

List of Program Requirements:

1. Menu screen that allows user to select options.
 - 1.1. Allow user to start a new game.
 - 1.2. Allow user to load a saved game.
 - 1.3. Allow user to load their own maps.
 - 1.4. Allow user to exit the game.
2. The game must contain a least one explorable and/or solvable maze.
 - 2.1. All maze(s) must be completeable.
 - 2.2. Mazes graphics will consist of ASCII.
 - 2.2.1. Different characters will be used to represent different objects.
 - 2.3. Maze must contain Walls
 - 2.3.1. Walls will be denoted by the character '■'.
 - 2.3.2. Some walls will be breakable.
 - 2.4. Maze must contain space where player can move.
 - 2.4.1. Space denoted by the character ' '.
 - 2.5. Game must contain a feature to play random mazes.
 - 2.6. Game must contain a feature to play pre-built mazes.
 - 2.7. Mazes must get harder as the game progresses.
 - 2.7.1. Level of maze increases with completion.
 - 2.7.2. Maze size must increase.
 - 2.7.3. More monsters to fight as maze level increases.
 - 2.8. Maze must have keys to collect in the maze to complete it denoted by '*'.
 - 2.9. Parts of the maze must be only visible if the player has visited that area.
3. The game must contain a player.
 - 3.1. Player must be able to move.
 - 3.1.1. w/a/s/d will be used to control the player.

- 3.1.1.1. Program must check if users input valid.
 - 3.1.1.2. Must support movement with caps lock on.
 - 3.1.2. There must be tiles players can and cannot walk on (collision detection).
- 3.2. Player must have a combat level.
 - 3.2.1. Killing a monster will grant experience points (exp).
 - 3.2.2. Killing a monster will remove it from the game.
- 3.3. Player must have Health points (HP).
 - 3.3.1. There must be items that restore HP.
 - 3.3.1.1. Items will appear on the floor of the maze.
 - 3.3.1.2. Item denoted by '+'.
- 3.4. Player must have an exp count to track current exp.
- 3.5. Player must be able to attack monsters.
- 3.6. If the player dies the save is deleted permanently.
- 3.7. Player will be denoted with the character '@'
- 4. The game must contain Enemies.
 - 4.1. Monsters must be able to move.
 - 4.1.1. They must contain primitive AI to track the player.
 - 4.1.1.1. Enemies will stop tracking player if too far.
 - 4.1.2. Different monsters must have different speeds
 - 4.2. There must be different kinds of monsters.
 - 4.2.1. Trolls, Giants, Witches, ect.
 - 4.2.1.1. Enemies will use the first letter of their race to represent them on the maze.
E.g 'T' for troll, 'G' for giant.
 - 4.3. Monsters must be able to attack the player.
 - 4.4. Monsters must have a combat level.
 - 4.4.1. Combat level of monster will be scaled with maze level.
 - 4.5. Monster must have health points (HP).
 - 4.6. Game must contain 'Boss Battles' where the player is challenged with a difficult enemy.
- 5. Attacking or being attacked by a monster will activate the combat screen.
 - 5.1. The combat screen will allow players to attack the enemy.
 - 5.2. The combat screen will contain an ASCII representation of each enemy.
- 6. Combat will be based off dice rolls.
 - 6.1. Virtual dice will be rolled to see whose attack hit.
 - 6.2. The range and amount of die is based off the player's skill and level.
 - 6.2.1. Players will begin with a standard die.
 - 6.2.2. The number of die a player has is based off their level.
 - 6.2.3. The range of a die is based off the number of enemies they have killed.
- 7. User must be able to create own maze map.
 - 7.1. Map created in text document (notepad).
 - 7.2. User should specify map location to load map.
 - 7.3. Program must check users map before playing to see if valid.
- 8. Game must be saveable in a fully recoverable state.
 - 8.1. Characters level and stats must be saved in files.
 - 8.2. The current maze the player is on must be save.
 - 8.3. If the player exits the game the game is automatically saved.
- 9. The game should be challenging.

Informal Program Specification:

Menu system -

- When the game is launched the user will be presented with a menu screen. This menu screen will allow the player to select options which will be controlled with a switch statement and enums via the input of the user.
- Loading a save from the menu will read the player's save file using fstream which contains information such as the player's level, HP and position. Then loading this information into the corresponding variables. Loading a save will also load the current map into an array and print it too the screen.
- Loading a player made map will use fstream to load their map into an array while simultaneously inspecting each character to check if they are valid by using if statements.

Maze –

- All mazes must be completable so any maze created by the user or a randomly generated maze will be checked with a shortest path algorithm to find a solution. Alternatively use a one pass algorithm to turn the maze into a graph and solve the graph using a breadth or depth first search however may not be necessary because maze size will be small.
- Different Tiles on the map will be represented with different ASCII characters this information will be stored in the class of each tile and passed in through the constructor on creation.
- Random mazes will be created with either a recursive backtracker, Randomized Prim algorithm or randomized Kruskal's algorithm.
- Prebuilt mazes will be designed by the creator using a text editor to provide variation for the user and will be loaded after each 5 levels.
- Difficulty of the maze must increase as mazes are complete. This will be achieved by increasing the size of the maze variables when creating the maze. Enemy's level will also be scale up with the increase of the maze level. This will be done by giving the monsters more health points and attack damage with the likelihood of more powerful enemies appearing and more frequently.

Player-

- A player class will be created to encapsulate all variable attributed to the player and inherit from the sprite class.
- A player must be able to move within the maze. This will be achieved using a switch statement with a user input to select the direction they wish to move in controlled by w/a/s/d. A function will then control the position variables of the player and move them accordingly. Capital letters variants of w/a/s/d will also be supported in the switch statement and all user input will be check for validity with exception handlers to ensure the program doesn't crash.
- A player must also be able to detect collision with an unmoveable tile. Once the user has selected the direction they wish to move in the movement will be check by a function to check for collisions. If a collision is detected then the player is not moved.
- A player must have a level attribute that controls the number of die they can roll during combat. This variable will be stored in the player class and the player's level is increased

by gaining exp from defeating enemies. Once an enemy has been defeated it will be removed from the vector of pointers and removed from the game.

- Player will have a health point's attribute that will be stored in the player class and will be decreased when attacked by an enemy. A player will be able to recover health by consuming health potions found on the maze floor.
- When the player dies the save file will be deleted permanently and the user will be returned to the menu. The save file will be over written with blank value using fstream and writing to the file.

Enemies -

- Enemies must populate the maze and will be created from a Monster class and will also inherit from the sprite class.
- Enemies must be able to move with a primitive AI. Every game tick an enemy on maze will decide whether to move, remain idle or attack. If a player is within the enemies vision the enemy will begin to follow the player by calculating the player's position with A* pathfinding. The enemy will use a basic AI to track the player. Once the player is out of view the enemy will stop following the player. The enemies will also move randomly around the maze. This will be done by selecting a random number from 1 to 4 and mapping that to a direction.
- A variety of enemies will be introduced into the game by having different initial values given to the constructor.
- All monsters will be: [C]yclops, [D]ragon, [G]iant, [g]oblin, [O]rc, [P]ixie, [S]pider, [s]keleton, [T]roll, [W]itch, [w]olf, [Z]ombie.
- If the enemy is adjacent to the player it will run the combat function.
- Monsters, just like the player, will have a level and dice values which will be initialized at run time when the monster object is created.
- Boss battles will be added via pre-made maps. They will use the standard Monster classes however will be much more powerful and more difficult to defeat.

Combat –

- During combat a random number generator will select a pseudo random number for both the monster and player to determine who attacks first based on the range of their dice.
- The attack screen will allow players and enemies to take it in turns rolling dice to attack one another. This will use random numbers between two values. The range of the numbers will be determined by a counter that increases based on the number of kills a player has.
- The ASCII representation of the monster will be pre created in a file and read in using fstream.

Saving –

- Saves must be fully recoverable and data should automatically save when a crash occurs

Comparison with Existing Programs:

Pac-man:

The maze game specified draws a lot of inspiration from the reviewed games. The grid based movement is very similar to Pac-man. Another functionality that will be akin to Pac-man is the Tracking AI that the enemies have. The player will be chased around the maze by enemies until the player is out of range or they enter combat. The proposed game differs in many ways, mostly just by having much more content than Pac-man. One of these features is the combat system. In pac-man you're able to eat enemies once a pac-pellet has been consumed, however in the suggested idea combat will be much more statistics and probability based with dice roll determining attacks. However another feature that differentiates its self from pac-man is its multiple unique level designs and random levels compared to pac-man's single level. Both however will be infinitely playable with the random map feature of the proposed game.

Labyrinth:

Labyrinth Allows players to create their maps and upload them so other players can play their creations. The suggested maze game takes inspiration from this by giving players the ability to create their own maps in a text editor and load that file into the game. This also means they are able to share their creations with friends. One similar feature that 'The world's hardest game' also has is a visible goal state which the user can see to determine how they are going to complete the maze before they even begin. The biggest difference between the two programs is probably the use of actuators in Labyrinth to control the ball

Dungeon Master:

Unlike Game Master the combat in the suggested game will be turn based with player and enemy taking it in turns to attack one another. Using the probability of dice rolls to determine attack damage rather than real time combat. An obvious difference is that Game Master looks 3 Dimensional with the player being able to move on a 2 Dimensional plane and a first person view. However the suggested game will have a top down 2 dimensional view. The proposed game will also only have the one player unlike Dungeon Master which allows the customisation of 4 characters in a team. One similarity is that both games will contain a large number of enemies to encounter with each having its own set of skills and difficulty. Both are also rouge likes so once the player dies the game ends with the saved delete making them both very challenging to complete. The game with the most similarities is Dungeon master.

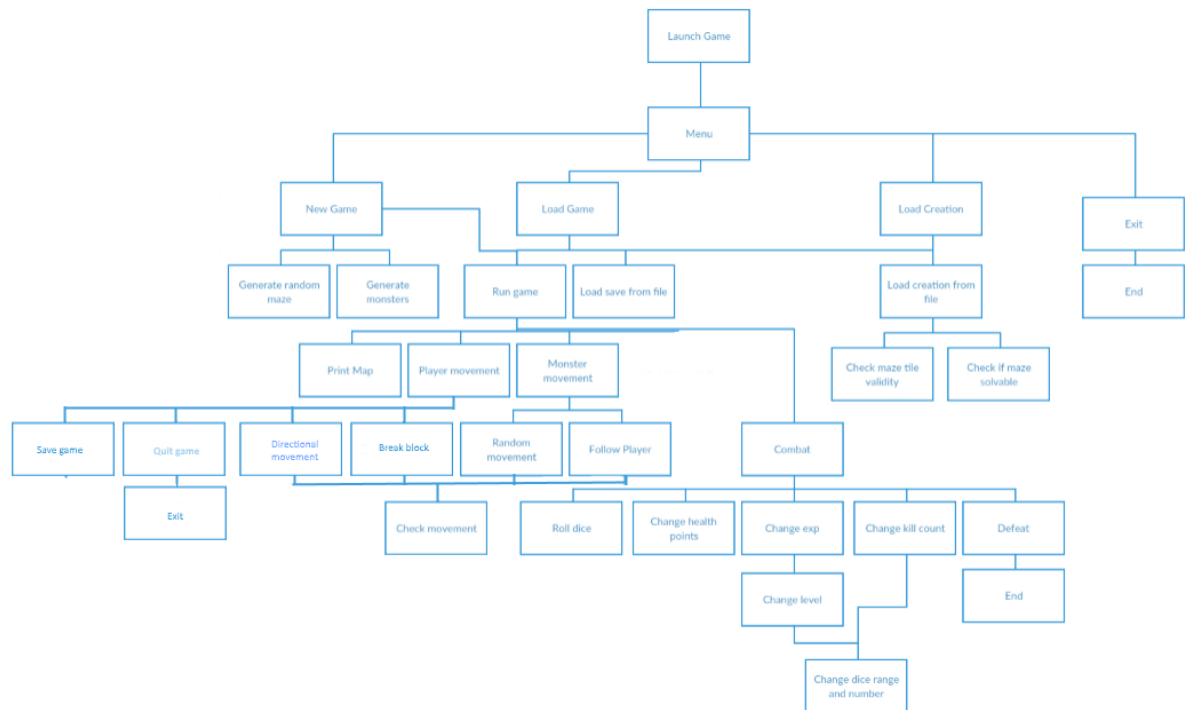
The Worlds hardest game:

The world's hardest game series of games is extremely challenging and takes persistence to complete. The suggested game will also strive to be very challenging for the player. This is will be achieved by another feature they have in common which is that with each successive competition of a maze the game gets harder.

Problem Decomposition:

Work breakdown structure:

Work breakdown structure shows a simplistic break down of the game as a whole covering the main features that is require by the game.



Features and their functions:

Class diagrams: UML diagrams

```
Class GameSystem
Public:
    GameSystem()
    ~GameSystem()

    void menu()
    void loadGame()
    void saveGame()
    void deleteSave()
    void run()

    void createMonsters()
    void createMaze()
    void drawMap()
    void CheckMove()
    void MoveEntities()
    void Combat()
Protected:
    bool _gameOver
    Tile _map[][]
    Vector <Sprite> Monsters
    Vector <int> xStack
    Vector <int> yStack
```

- **Void menu()** – Prints options to the console and allows user to select which option they like via getchar(). A switch is then used to run the according function.
- **Void loadGame()** – Loads current maze map into a 2D array from a file using an ifstream and for loops to control in which location the map tiles go. A separate file is loaded to collect variable values to recreate the sprites within the game by saving them into the vector of sprites.
- **Void saveGame()** – Saves the game using ofstream to a maze file which saves the current maze and a player and entities file which save all entities. Save can be selected by the user or is run whenever the game crashes.
- **Void deleteSave()** – Deletes all save information about a level using ofstream. Called when the player is defeat.
- **Void Run()** – Runs the main game loop and allows for the printing of the map and sprite movement.
- **Void createMonster()** – Creates monsters and places them on the map. Uses players level and maze level to determine difficulty of monsters.
- **Void createMaze()** – Creates a randomized maze using a maze creation algorithm.
- **Void drawMap()** – Draws the map from the array using for loops and prints it to the console.

- **Void checkMove()** – checks a player's movement and checks for collision using a switch statement to determine if it is a solid objects, health or enemies. And will take the modified X,Y co-ordinates and a pointer to a player for parameters.
- **Void moveEntities()** – Once an entity move has been checked it will change its position co-ordinates. Takes changed X and Y co-ordinates and moves the entities to their new locations in `_map[][]`.
- **Void combat()** – will initiate combat between a monster and player and will take one of each as parameters. The combat function will then roll dice between both entities to decide who wins.
- **Bool _gameOver** – Controls the main game loop. If a player die `_gameOver` is set to true.
- **Tile _map[][]** – A 2D array of tile objects which stores the information about the map. Used to print the map to the screen and holds information about en
- **Vector <Sprites> Monsters** - a vector of sprites that will hold all entities within the game and be used to move sprites around the screen.
- **Vector <int> xStack** – keeps track of X co-ordinate of visited notes when creating a random maze.
- **Vector <int> yStack** – keeps track of Y co-ordinate of visited notes when creating a random maze.

```

Class Tile
Public:
    Tile()
    ~Tile()

    char getTileChar()
    bool getTileSolid()
protected:
    char _tileChar
    bool _tileSolid

```

```

Class HPTile: Public Tile
Public:
    HPTile()
    ~HPTile()

    int getHPTileHeal()
    bool getUsed()
Protected:
    int _HPHeal()
    bool _used()

```

Class Tile:

- **Char getTileChar()** – gets the character associated with the specific tile and returns it.
- **Bool getTileSolid()** – gets whether the entity is able to move into that space or not and returns true or false accordingly.
- **Char _tileChar** – holds the character of the tile that was passed in through the constructor.
- **Bool _tileSolid** – holds true or false value for whether an entity can move to a space.

HP inherited tile:

- **Int getHPTileHeal()** – gets the amount the health heals and returns it.
- **Bool getUsed()** – gets whether the health points have been used or not.
- **Int _HPheal** – holds the value for how much the Health point heal
- **Bool _used** – holds true or false value for whether the health points have been used or not

```

Class Sprite
Public:
    Sprite()
    ~Sprite()

    void getSpritePos()
    void setSpritePos()
    char getSpriteChar()
    void setSpriteChar()
    bool getSpriteSolid()
    Tile getTileOn()
    void setTileOn()
    int getHP()
    void setHP()
    int getLevel()
    void setLevel()
    int rollDice()
    void setDice()
Protected:
    int _xPos
    int _yPos
    char spriteChar
    bool spriteSolid
    char _tileChar
    Tile _tileOn
    int _level
    int _HP
    Dice _dice

```

```

Class Player: Public Sprite
Public:
    Player()
    ~Player()

    int getMazeLevel()
    void setMazeLevel()
    int getExp()
    void setExp()
    int getKillCount()
    void setKillCount()
protected:
    int _mazeLevel
    int _killCount
    int _exp

```

```

Class Monster: Public Sprite
Public:
    Monster()
    ~Monster()

    void randomMove()
    void followPlayer()
Protected:

```

Class Sprite:

- **Void getSpritePos()** – gets the sprite's X and Y position by passing in local variables as pointers.
- **Void setSpritePos()** – sets the sprite's X and Y position by passing it new values
- **Char getSpriteChar()** – gets the sprites Ascii symbol
- **Void setSpriteChar()** _ sets the sprite ascii symbol
- **Bool getSpriteSolid()** – gets whether a sprite is solid or not
- **Tile getTileOn()** – gets the Tile the entity is currently on top of and returns it. Used to modify the map to original state when entity moves.

- **Void setTileOn()** – sets the Tile the entity is currently stood on.
- **Int getHP()** – gets the current HP value of the entity and returns it.
- **Void setHP()** – sets the HP value to a new value, usually when taking damage in combat.
- **Int getLevel()** – gets the level of the entity and returns it.
- **Void setLevel()** – sets the level of the entity.
- **int rollDice()** – getter function that rolls the dice of an entity, adds up the dice rolls and returns the value as an integer.
- **Void setDice()** – sets the value of dice. When a player levels up or reaches a specific killCount dice number and range increase accordingly.
- **Int _xPos** – Holds the X position of the entity.
- **Int _yPos** – Hold the Y position of the entity.
- **Char _spriteChar** – Holds the sprite's Ascii symbol
- **Bool _spriteSolid** – Holds the value if the sprite is solid or not
- **Char _tileChar** – Holds the ASCII character that represents the entity.
- **Tile _tileOn** – Holds the tile that the entity is currently stood on.
- **Int _level** – hold the current level of the entity.
- **Int _HP** – holds the current health points value of the entity.
- **Dice _dice** – holds the dice value that contains the range and number of dice.

Player inherited class from sprite:

- **Int getMazeLevel()** – gets the level of maze the player is on.
- **Void setMazeLevel()** – sets the level of the maze the player is on.
- **Int getExp()** – gets the current exp value of the player and returns it.
- **Void setExp()** – sets the new value of the exp. Usually after killing an enemy exp is rewarded.
- **Int getKillCount()** – gets the number of enemies the player has killed. Used to increase the range of the players dice.
- **Void setKillCount()** – sets the number of enemies killed to the new value. Increases by one when any enemy is defeated.
- **int mazeLevel** – holds value of current maze level.
- **Int killCount** – keeps the number of enemies defeated by the player.
- **Int exp** – keeps the amount of exp the player has.

Monster inherited class from sprite:

- **Void randomMove()** – random move is the default movement of Monster entities it will use a random number generator to determine 1 of 5 states, Left, right, up, down and idle states.
- **Void followPlayer()** – will use A* path finding to track the player when they are in the vicinity.

```

Class Dice
Public:
    Dice()
    ~Dice()

    void setDice
Protected:
    int _minRange
    int _maxRange
    int _num

```

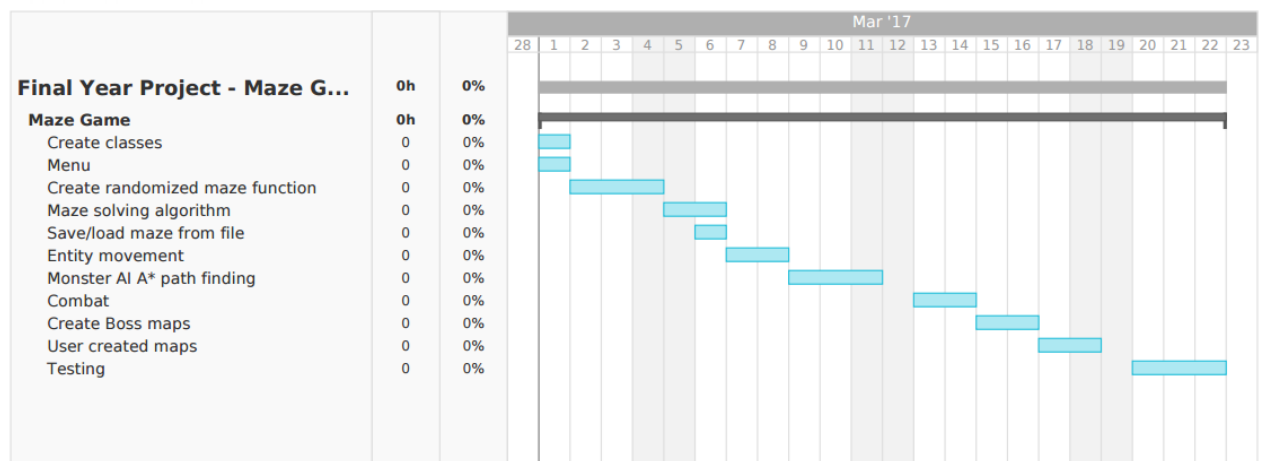
Class Dice:

- **Void setDice()** – sets the value of the dice using range and number of dice parameters passed to it.
- **Int _minRange** – holds the minimum range of the dice the entity has.
- **Int _maxRange** – holds the maximum range of the dice the entity has.
- **Int _num** – holds the number of dice you have.

Time scheduling:

Gantt chart shows the main/most difficult features of development and implementation with a rough timeline of when they will be completed. The Gantt chart allowed me to identify some of the more vital parts of the program and which bits I should concentrate on first.

 Created with Free Edition



High-Level Flowchart / Pseudocode:

Pseudo code:

Creating Random mazes: Recursive backtracker

```
#####
#.#.#.#.#.#.#.#.#.#
#####
#.#.#.#.#.#.#.#.#.#
#####
#.#.#.#.#.#.#.#.#.#
#####
#.#.#.#.#.#.#.#.#.#
#####
#.#.#.#.#.#.#.#.#.#
#####
#.#.#.#.#.#.#.#.#.#
#####
#.#.#.#.#.#.#.#.#.#
#####
#.#.#.#.#.#.#.#.#.#
#####
```

Representation of a 10x5 maze that the function will use to calculate a random maze.

```
First cell becomes the current cell and mark it as a visited cell
While there are unvisited cells
  If the current cell has any neighbours 2 spaces away which have not been visited
    Generate random number
    Choose unvisited neighbour based on random number
    Push the current cells X and Y co-ordinates to vector
    Remove the wall between the current cell and the chosen cell
    Make the chosen cell the current cell and mark it as visited
  Else if stack is not empty
    Pop a cell from the stack
    Make it the current cell
  Else
    Maze complete
  End
End
```

Loading map from file:

Simple abstraction for reading a file. Process will be more complex to store other attributes of the tile such as whether it is solid and if it is visible or not.

```
open file for reading
for 0 to width
  for 0 to height
    get character from file
    case based on character
      case # then create wall object and add to 2D array
      case . then create air object and add to 2D array
      case + then create HP Object and add to 2D array
      case * then create star Object and add to 2D array
    end
  end
end
close file
```

Movement for players:

Uses a switch statement to determine the users input which moves it to the required function.

```
get movement input from user
get player x,y co-ordinates
case based on input
    case W/w then checkmove(x, y - 1)
    case D/d then checkmove(x + 1, y)
    case D/s then checkmove(x, y + 1)
    case A/a then checkmove(x - 1, y)
    case P/p then breakblock()
    case M/m then savegame()
    case N/n then quit()
end
```

Check entity movement (checkMove()):

Checks the tile the player wants to move to before moving the player. If the tile is not solid the player will be moved. The player will also be able to interact with other objects within the maze.

```
if map[y][x] tile is not solid then
    then current tile becomes the tile stood on by player
    player tile stood on becomes new tile
    set player position to X and Y values
else
    player will not move
end

if map[y][x] tile is a '*' then
    player's star count + 1
end

if map[y][x] tile is a '+' then
    increase players health by 25
end

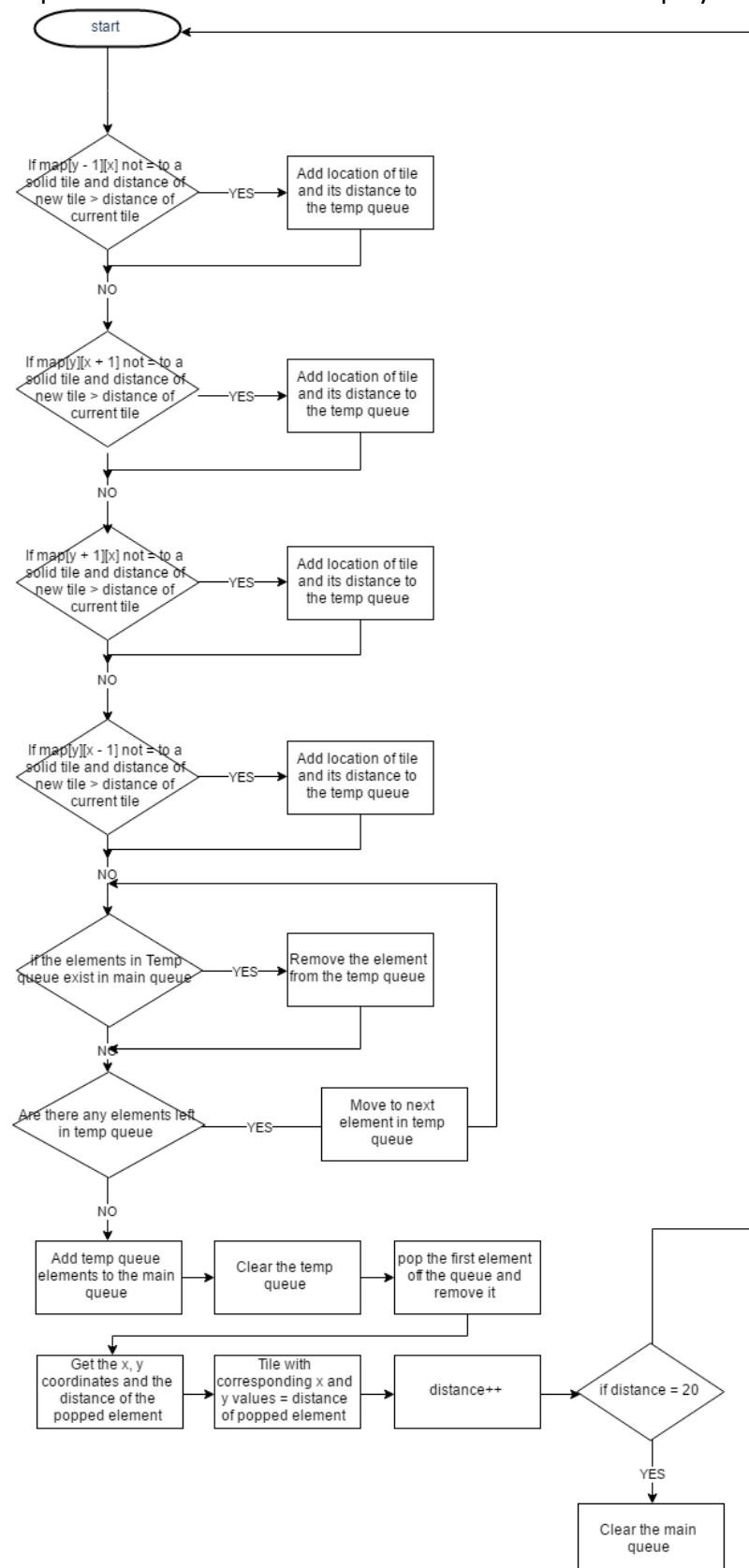
if map[y][x] tile is a '1' then
    increase map size
    reset the star count
    generate a new maze
    move player to map[1][1]
end

if map[y][x] tile is a letter then
    for i = 0 to number of monsters
        if monster(i) X and Y = player's X and Y then
            combat(player, monster(i))
        end
    next
end
```


Flow charts:

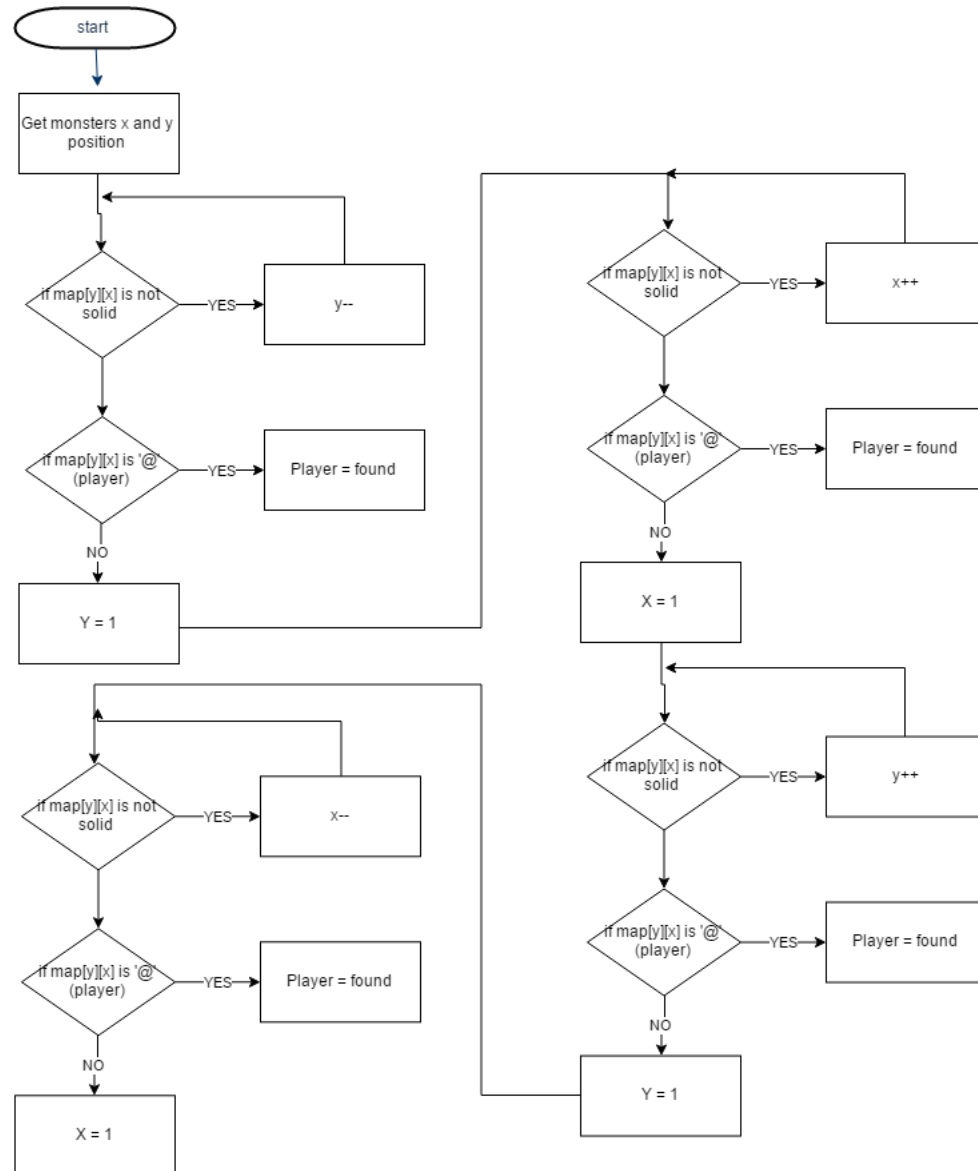
Distance map:

Calculating the distance map from the player. Uses a breadth first search method to explore the maze and calculate how far it is from the player.



Monster vision:

Calculates whether the players has been seen by the monster this turn. Uses a depth first search to look for the player.



Development

The requirements specified in the original design document have been used to identify which features have been added to the maze game and was updated periodically to see progression. The requirements were used regularly to make sure the project remained on topic and within the specification of the design document. The development log below goes into more depth on the features implemented and explains why some features were removed and why others have been uncompleted.

■	Completed feature
■	Uncompleted feature
■	Removed feature

- 9.1. Allow user to start a new game.
- 9.2. Allow user to load a saved game.
- 9.3. Allow user to load their own maps.
- 9.4. Allow user to exit the game.
- 10. The game must contain a least one explorable and/or solvable maze.
 - 10.1. All maze(s) must be completeable.
 - 10.2. Mazes graphics will consist of ASCII.
 - 10.2.1. Different characters will be used to represent different objects.
 - 10.3. Maze must contain Walls
 - 10.3.1. Walls will be denoted by the character '■'.
 - 10.3.2. Some walls will be breakable.
 - 10.4. Maze must contain space where player can move.
 - 10.4.1. Space denoted by the character ' '.
 - 10.5. Game must contain a feature to play random mazes.
 - 10.6. Game must contain a feature to play pre-built mazes.
 - 10.7. Mazes must get harder as the game progresses.
 - 10.7.1. Level of maze increases with completion.
 - 10.7.2. Maze size must increase.
 - 10.7.3. More monsters to fight as maze level increases.
 - 10.8. Maze must have keys to collect in the maze to complete it denoted by '*'.
 - 10.9. Parts of the maze must be only visible if the player has visited that area.
- 11. The game must contain a player.
 - 11.1. Player must be able to move.
 - 11.1.1. w/a/s/d will be used to control the player.
 - 11.1.1.1. Program must check if users input valid.
 - 11.1.1.2. Must support movement with caps lock on.
 - 11.1.2. There must be tiles players can and cannot walk on (collision detection).
 - 11.2. Player must have a combat level and skills.
 - 11.2.1. Killing a monster will grant experience points (exp).
 - 11.2.2. Killing a monster will remove it from the game.
 - 11.3. Player must have Health points (HP).

- 11.3.1. There must be items that restore HP.
 - 11.3.1.1. Items will appear on the floor of the maze.
 - 11.3.1.2. Item denoted by '+',
- 11.4. Player must have an exp count to track current exp.
- 11.5. Player must be able to attack monsters.
- 11.6. If the player dies the save is deleted permanently.
- 11.7. Player will be denoted with the character '@'
- 12. The game must contain Enemies.
 - 12.1. Monsters must be able to move.
 - 12.1.1. They must contain primitive AI to track the player.
 - 12.1.1.1. Enemies will stop tracking player if too far.
 - 12.1.2. Different monsters must have different speeds
 - 12.2. There must be different kinds of monsters.
 - 12.2.1. Trolls, Giants, Witches, ect.
 - 12.2.1.1. Enemies will use the first letter of their race to represent them on the maze.
E.g 'T' for troll, 'G' for giant.
 - 12.3. Monsters must be able to attack the player.
 - 12.4. Monsters must have a combat level.
 - 12.4.1. Combat level of monster will be scaled with maze level.
 - 12.5. Monster must have health points (HP).
 - 12.6. Game must contain 'Boss Battles' where the player is challenged with a difficult enemy.
- 13. Attacking or being attacked by a monster will activate the combat screen.
 - 13.1. The combat screen will allow players to attack the enemy.
 - 13.2. The combat screen will contain an ASCII representation of each enemy.
- 14. Combat will be based off dice rolls.
 - 14.1. Virtual dice will be rolled to see whose attack hit.
 - 14.2. The range and amount of die is based off the player's skill and level.
 - 14.2.1. Players will begin with a standard die.
 - 14.2.2. The number of die a player has is based off their level.
 - 14.2.3. The range of a die is based off the number of enemies they have killed.
- 15. User must be able to create own maze map.
 - 15.1. Map created in text document (notepad).
 - 15.2. User should specify map location to load map.
 - 15.3. Program must check users map before playing to see if valid.
- 16. Game must be saveable in a fully recoverable state.
 - 16.1. Characters level and stats must be saved in files.
 - 16.2. The current maze the player is on must be save.
 - 16.3. If the player exits the game the game is automatically saved.
- 17. The game should be challenging.

Development log

Completed feature
Uncompleted feature
Removed feature

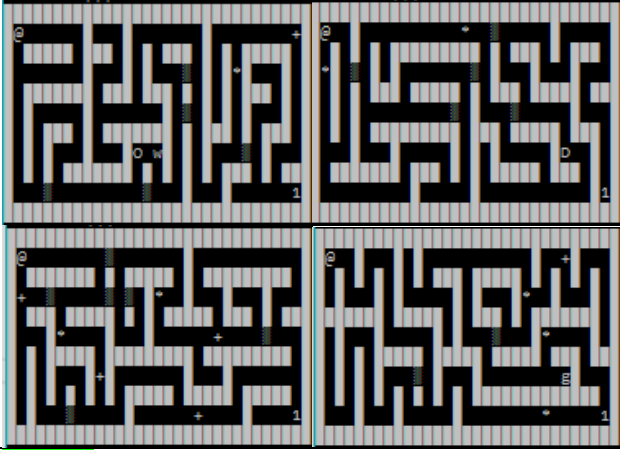
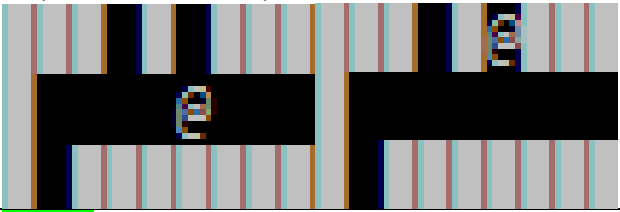
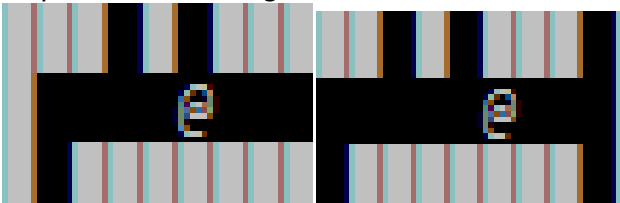
Date	Working on	Description
01/03/17	Classes	Implementing the pre-designed classes with all their getters and setters to make them functional once the main program is written.
01/03/17	Menu	Created a simple menu using a switch statement that allows the user to select the options 'new game', 'load game' or 'exit'. Added error handling to make sure users could only select the three options. The switch statement will then activate the required function
02/03/17	Initialised monsters within the game	Created the initialisers for all monsters within the game. Consisted of balancing monster strengths with health and damage.
02/03/17	Random maze generation	Created an algorithm that generated a random maze which was based off primms algorithm. Within the algorithm was added random spawns for stars, health and monsters. Wall tiles were made solid so players and monsters within the maze are not able to pass through them. Random maze generation used a checkboard grid to create the effect of walls within the maze.
02/03/17	Map printing	Printing the map to the console was added using cout.
04/03/17	Entity movement	Added the ability for players and monsters to move around the maze. Movement is checked for solid objects like walls and prevents the entity from moving. It also checks if the player encounters objects such as a star, health or the exit of the map. Used switch statements for both player and monster movement
04/03/17	Block breaking	Block breaking was added so the player can break specific blocks within the level. The breakable block is replaced with an air tile to make it look removed.
04/03/17	Direction of player	In order to make block breaking more intuitive. The direction of the player was added. The last movement the player makes is the direction they are facing.
05/03/17	Validating player movement	Validation was added to the player's turn to make sure they selected valid options. This was achieved by using if statements to only allow specific inputs.
06/03/17	Monster vision	Created an algorithm that searches for the player each turn in the 'line of sight' of the monster. The search uses a depth first search to search each path in all four directions. The monsters player found attribute will activate if the player is

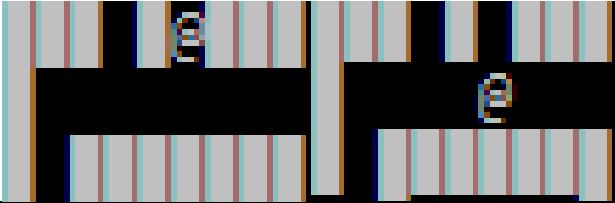
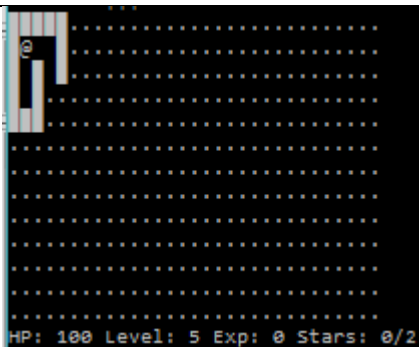
		seen, this will later be used to allow the monster to follow the player.
07/03/17	Player vision	Added vision to the player by setting only the tiles the player has seen in its line of sight to visited. Only the visited tiles were then printed to the screen. The rest were printed as blank spaces. Paths in all four directions are searched until a solid block is found and set to visited along with the two tiles either side of it to display the walls.
09/03/17	Distance map queue	Created a queue object which stores three values. X position of a tile, its y position and the distance the tile is from the player.
09/03/17	Monster AI	A simple breadth first search algorithm was used to calculate the distance from tiles to the player in a radius of 20 tiles. If a monster has seen the player the monster will begin to move closer and closer following decreasingly lower numbers to the player depending on the monsters speed.
15/03/17	Removed pre-built mazes and boss battles feature	Decided to remove the user built mazes from the game. Along with pre-built mazes with boss battles. This was because they would add a lot of time to development and didn't add much to the complexity or entertainment of the game.
15/03/17	Removed the ability for users to select their own mazes to be player from the menu	Longer including player built mazes with the game so therefore removed it from the menu system.
16/03/17	Combat	Added combat between two entities. Two entities passed into combat functions and use their dice values to randomly generate attack damage to see who wins. Loser is removed from the game. Winner receives exp.
16/03/17	Monster engage combat with player	Discovered monster is not able to engage in combat with player. The player is not stored in the monsters vector so monster is unable to access the player's information to battle it.
17/03/17	Removed ascii representation of each monster	No longer implementing ascii representation of each monster as would require a lot of file storage and time could be spend developing more important aspects of the game.
17/03/17	Removed combat between two monsters	Removed the ability for two monsters to engage in combat as a lot of monsters would die before the player could reach them making it very easy. Only player and monsters now able to enter combat.
18/03/17	Removed monster scaling with maze level	Removed monsters scaling with maze level. As maze size increases more monsters spawn which achieves the increase in difficulty desired.

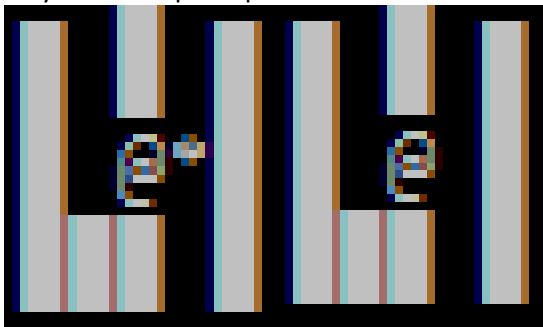
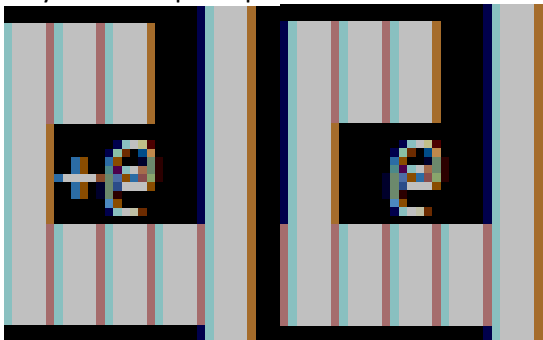
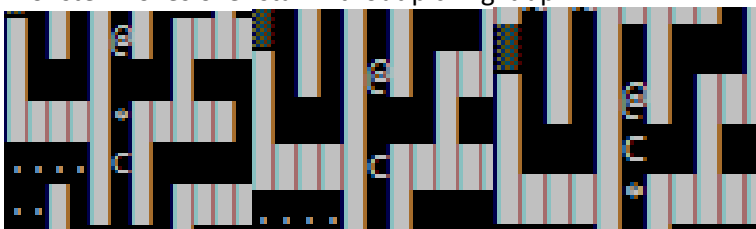
20/03/17	GUI	Added a GUI which displays the player's health, level, exp and stars collected and left to collect within the map.
23/03/17	Saving the game	Added the ability to save the game. Stored map data in one file and sprite data in another. Used CSV format to separate data elements.
24/03/17	Removed saved quitting feature	Game no longer saves when player quits the game.
25/03/17	Loading the game	Can now load the data that had been previously saved using fstream. Data is used by constructors to create the assets of the loaded game.
25/03/17	Overwriting saves (new game)	If a user wishes start a new game and currently has a saved game the system will notify that they are about to overwrite a saved game and once again asks if they wish to start a new game. If they select a new game the old saved game is deleted.
26/03/17	Deleting saves	When a player dies the save they currently have is deleted permanently by overwriting the saves with nothing.
27/03/17	Exiting the game	When exiting the game the system now asks if definitely want to exit as any unsaved data will be lost.
29/03/17	Finishing details	Added aesthetic details such as an Ascii menu system and an Ascii death message. These make the game feel more complete.
30/03/17	Vision dots	From the sample group that received the program said one thing they disliked was that occasionally the map would look completely explored however some paths had not been visited. '.'s were added to unvisited tiles to make it much easier for the user to see which paths had not been visited.
30/03/17	Dragons spawns	Another thing the sample group said was that dragons spawn too often. Decreased the chance of dragons spawning
31/03/17	Monster spawns	The sample group also noticed that monsters only spawn around the centre of the map. Changed the locations monsters can spawn now more spread out.
31/03/17	Monsters too hard	Decreased the difficulty of monsters

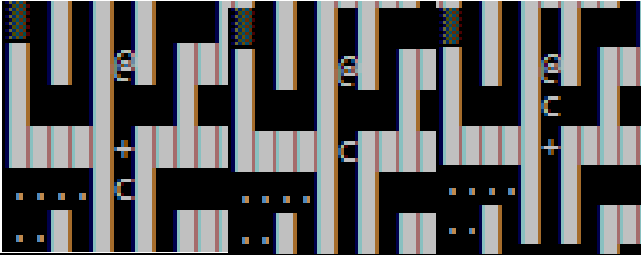
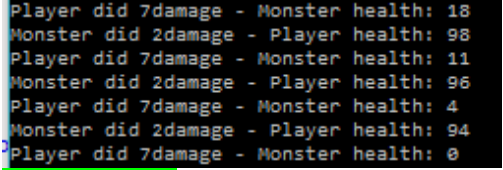
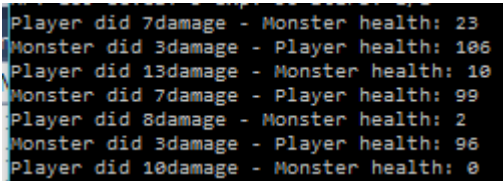
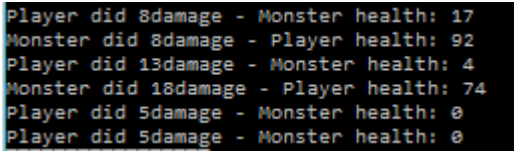
Test first development

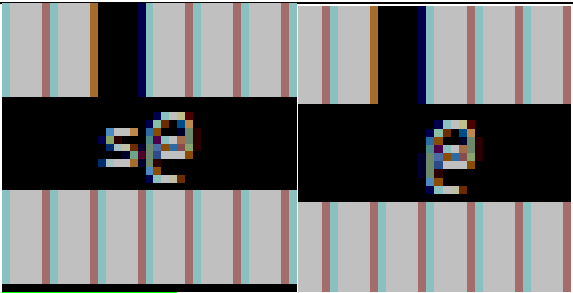
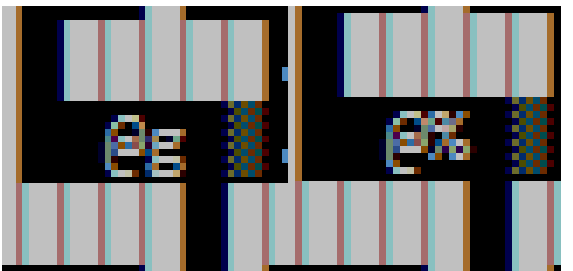
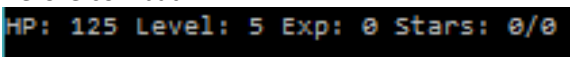
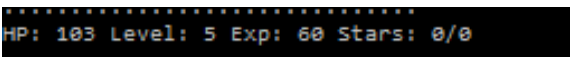

The test plan was written before the program was written. These tests were based off the initial requirements from the design document and represent a test first development method.



Test num	Description	Test data	Expected output	Actual output
1a	User must not be able to enter an invalid input into the console	'1'	New game will start and game loop begins	PASSED New game created as expected and game loop begins
1b		'2'	Load previous game and start game loop	PASSED The previously saved game was loaded and game loop begins
1c		'3'	Exit	PASSED Game is exited and application is closed
1d		Any other input	Refresh menu	PASSED Menu is refreshed and no other affect occurs
2	Generated maze must be random	Running the application multiple times to test randomness	A random maze will be displayed on screen	PASSED Maze generation run 4 times and appears to be random 
3a	User input must be validated on player turn	'w', 'W'	Player will move up	PASSED Player able to move up 
3b		'a', 'A'	Player will move right	PASSED Player able to move right 

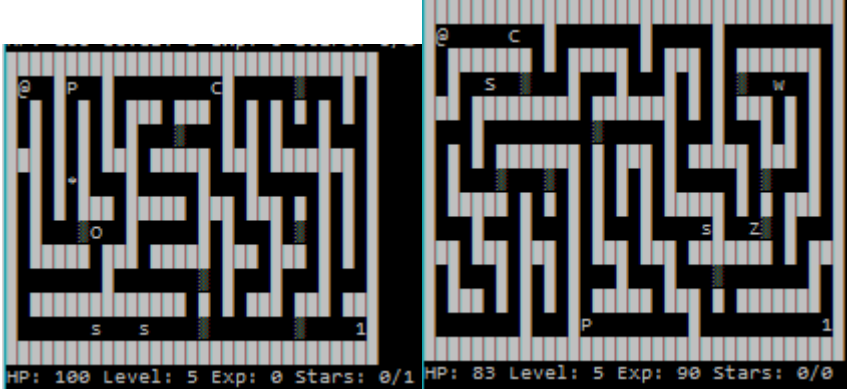

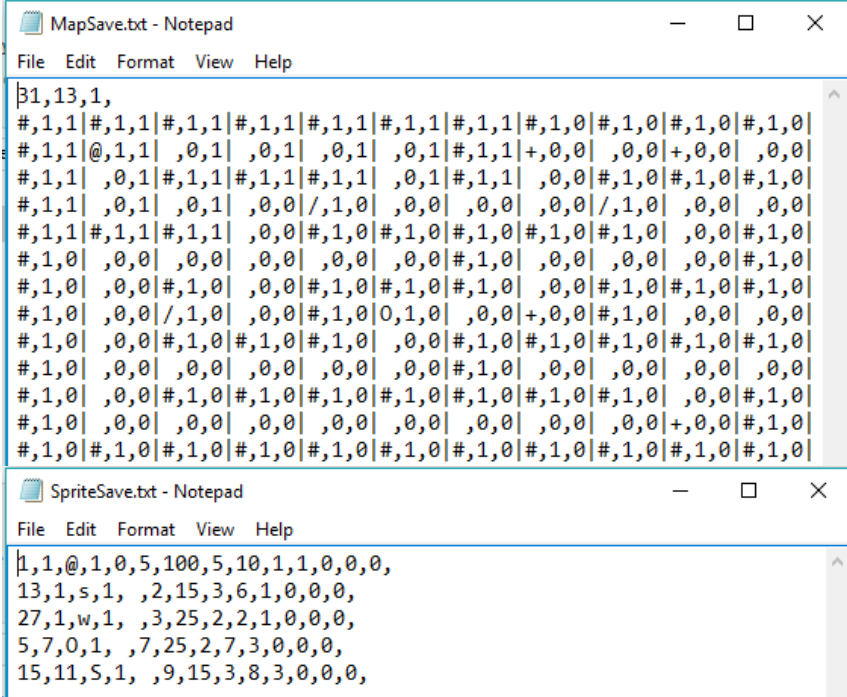
3c		's', 'S'	Player will move down	<p>PASSED</p> <p>Player able to move down</p> 
3d		'd', 'D'	Player will move left	<p>PASSED</p> <p>Player able to move left</p> 
3e		'p', 'P'	Player will break wall	<p>PASSED</p> <p>Player able to break wall</p> 
3f		'm', 'M'	Game will be saved	<p>PASSED</p> <p>Game saved</p> 
3g		'n', 'N'	Game will be quit	<p>PASSED</p> <p>Game asks you to quit and then quits</p> 
3h		Any other input	no action will be taken	<p>PASSED</p> <p>No action taken and input is ignored</p>
4	Player created correctly	Create a game	Player should be able to move and with health, stars load correctly	

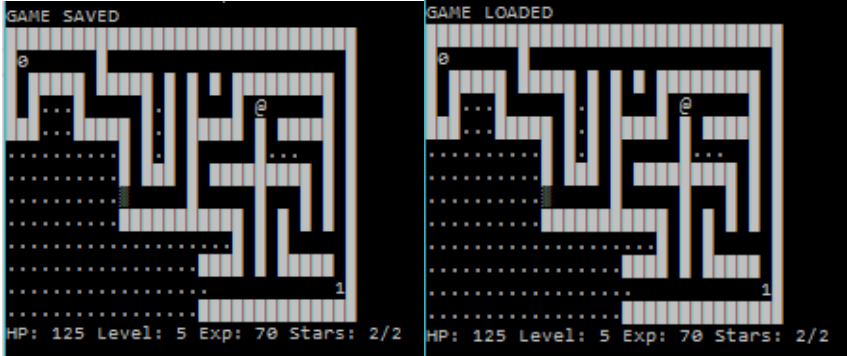
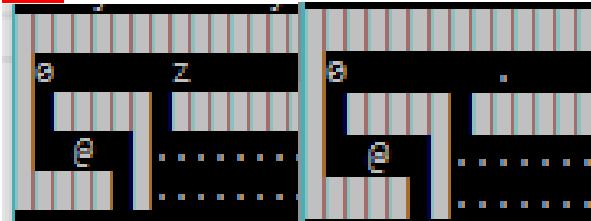

				<div>Autos</div> <table><tr><th>Name</th><th>Value</th><th>Type</th></tr><tr><td>player</td><td>{_mazeLevel=1 _killCount=0 _playerDirection=1 }</td><td>Player</td></tr><tr><td> Sprite</td><td>{_posX=1 _posY=1 _spriteChar=64 '@' ...}</td><td>Sprite</td></tr><tr><td> _mazeLevel</td><td>1</td><td>int</td></tr><tr><td> _killCount</td><td>0</td><td>int</td></tr><tr><td> _playerDirection</td><td>1</td><td>int</td></tr></table>	Name	Value	Type	player	{_mazeLevel=1 _killCount=0 _playerDirection=1 }	Player	Sprite	{_posX=1 _posY=1 _spriteChar=64 '@' ...}	Sprite	_mazeLevel	1	int	_killCount	0	int	_playerDirection	1	int
Name	Value	Type																				
player	{_mazeLevel=1 _killCount=0 _playerDirection=1 }	Player																				
Sprite	{_posX=1 _posY=1 _spriteChar=64 '@' ...}	Sprite																				
_mazeLevel	1	int																				
_killCount	0	int																				
_playerDirection	1	int																				
5a	Sprite collision with solid objects	Test player walking into solid object	Player will not be able to occupy the same tile as a solid object	<div>PASSED</div> <div>Player not able to occupy the same tile as solid object</div>																		
5b		Observe monster sprites to see if they are able to walk into walls	Monsters will not be able to occupy the same tile as a solid object	<div>PASSED</div> <div>Monster not able to occupy the same tile as solid objects</div>																		
6a	Collectables must be obtainable by players	Player moves onto '*' tile	Player picks up collectable, it is removed from the map and player star count is increased by 1	<div>PASSED</div> <div>Player able to pick up star</div> 																		
6b		Player moves onto '+' tile	Player picks up collectable, it is removed from the map and player health increases by 25	<div>PASSED</div> <div>Player able to pick up health</div> 																		
7a	Collectables must not be obtainable by monsters	Monster moves onto '*' tile	Monster moves over tile without removing it from the map	<div>PASSED</div> <div>Monster moves over star without picking it up</div> 																		

7b		Monster moves onto '+' tile	Monster moves over tile without removing it from the map	PASSED Monster moves over health without picking it up 
8a	Entering combat	Player moving onto the same tile as a monster	Player will enter combat with the monster	PASSED Player able to engage in combat
8b		Monster moving onto the same tile as player	Monster will enter combat with the player	FAILED Monsters unable to enter combat with player ISSUE: Monsters cannot 'see' player as no way of accessing the players attributes to engage in combat.
8c	Combat damage must be random each turn	Player and monster engage in combat	Player and monster damage will be random each turn	FAILED Damage between monster and player was not random  FIXED: PASSED Srand() was being seeded every turn which caused the same value to be chosen for the seed. Srand() now seeded once at the start of the program 
9	Player and monster health must change after each turn	Player and monster engage in combat	Player and monster health should decrease after each turn	PASSED Player and monsters lose health after each turn 
10	Killing a monster will cause corpse to appear	Attacking a monster and killing it	Tile where monster was killed will now display a '%' on the map	FAILED

				 <p>FIXED: PASSED</p> <p>▶ <code>_tileOn</code> {_tileChar=0 '\0' _tileSolid=false _visited=true ...}</p> <p>Some monsters have newline character for tileOn</p> 
11	Killing a monster will grant exp to player	Player killing a monster	Player's exp should increase up the GUI	<p>PASSED</p> <p>Before combat:</p>  <p>After combat:</p> 
12	Player being killed will delete the current save game	Player being killed by a monster	Saved should be delete in the file	<p>PASSED</p> <p>Saved game stored in file is deleted correctly</p> 
13	Monsters able to see player when they are in direct line of sight and follow the player	Move player into line of sight of monster	Monster should 'see' player and start to follow player at monsters movement speed	<p>PASSED</p> <p>Monster 'sees' player and begins to follow the player around at set movement speed</p>
14	Monsters will move around randomly when player is not located	N/A	Monster should move randomly around the map while the player	<p>PASSED</p>

			is not located	
15a	Breadth first search calculates distance map correctly	N/A	Console should print a map that spreads 19 tiles away from the player	<div>PASSED</div>  <p>Map correct in all directions</p>
15b	Monsters stop following player if more than 20 tiles away and return to random movement	Player out of range of monster (this applies to monsters who move slower than the player)	Monster should return to a random movement once the player is out of range	<div>FAILED</div>  <p>Monster stops following player however remains idol</p>
16	Player finishes level when move over finish tile	Player moves over '1' tile	A new maze should be created with new monsters with the player starting at '0'	<div>PASSED</div> <p>A maze is created with new monsters and the player is moved to the starting location of '0'</p>
17	Map size increase after level complete	Player completing a level	Maze should be 4 tiles bigger in the X direction and 2 in the Y	<div>PASSED</div> <p>Maze increased in size correctly</p>

				
18	Players vision of map increases as player line of sight changes.	Move player around map	More map should be revealed as more is explored	<p>PASSED</p> <p>More map appears as player explores the maze</p> 
19	Game saves to file correctly	Make a saved game while playing the game	Saved information should be stored in a file correctly	<p>PASSED</p>  <p>Information appears to be correct in both files</p>

20	Game loaded correctly from file	Load the game in the menu	Map should be loaded with all previous assets correctly	<div><div>PASSED</div><div></div><div>All assets loaded correctly: health, exp, stars collected correct. Vision map correct, player position, size of map ect.</div><div>BUG:</div><div></div><div>When reloading a save monsters appear invisible and the tile they are on is marked as unvisited.</div></div>
21	Game not load if there is no saved game	Select load game without a save existing	Game prints message to with the message "No previous save game"	<div><div>PASSED</div><div>Prevents program from crashing</div><div></div></div>

Black box testing: - Exploratory testing

The testing log consists of bugs and issues come across during and after development of the program when play testing the program. The bug log consists of an issue that has been detected and a screenshot of the issue. If resolved a explanation is given.

Fixed problem
Unable to identify problem
Unfixed problem

I think the dragon fighting the goblin counted as the player in combat somehow.

Fixed: Only allow player to enter combat.



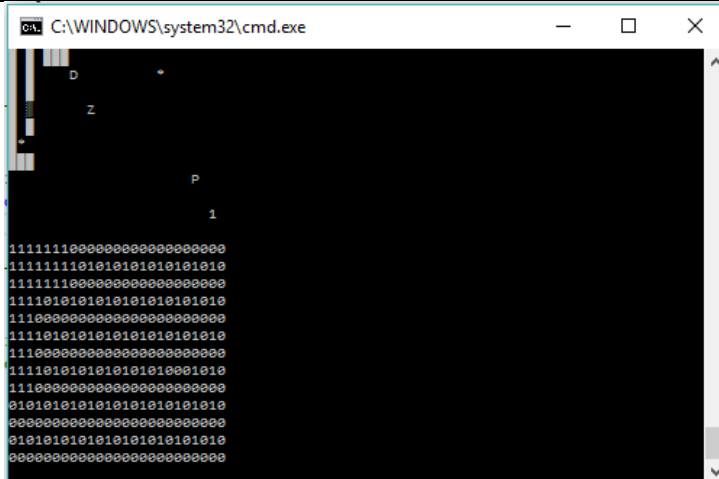
Enemy can spawn on 1,1 which cause the '0' to disappear

Fixed: don't allow monsters to spawn near 1,1. Changed the locations at which things can spawn.



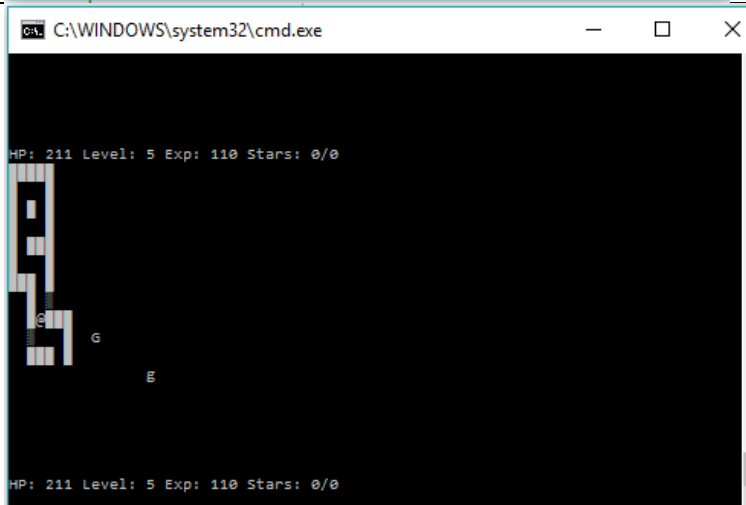
Below is the visited map.
For some reason every other row
alternates visited on tiles
might explain why some artefacts left
even when unvisited.

Fixed: reset visited after creating the
maze.



Sometimes monsters appear even when
tile should not be visible


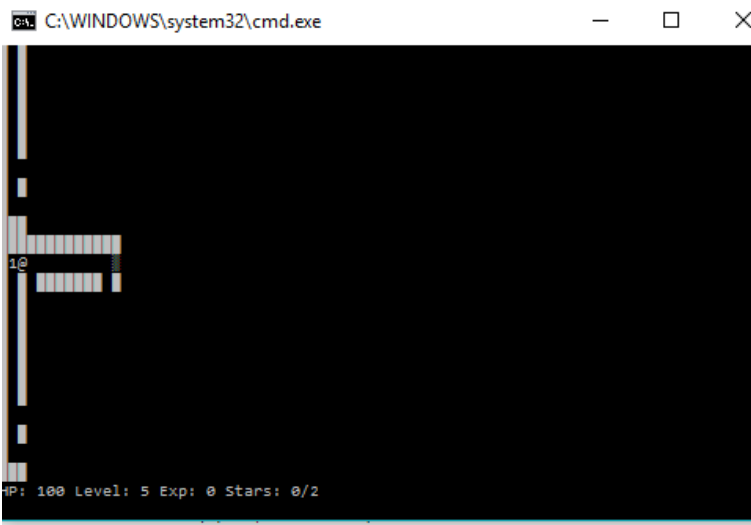
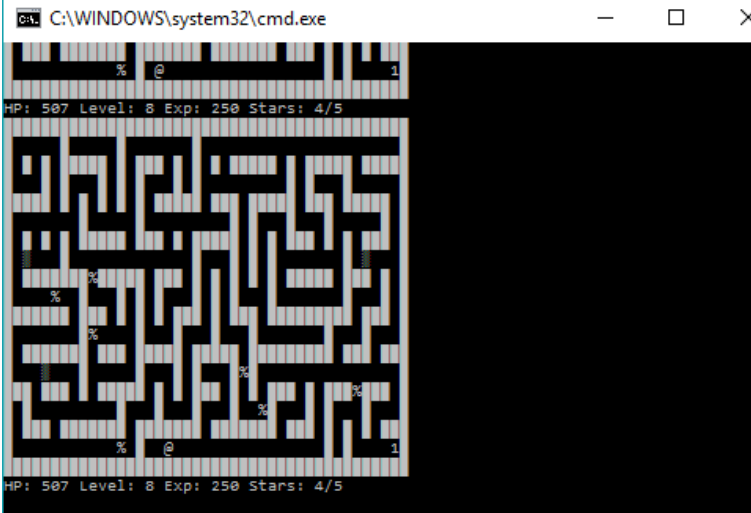
FIXED: Monsters not move when player
not seen



On level 2 the starting '0' has
disappeared – '1' still remains.

Fixed: Made sure to change the players
TileOn to '0' before moving it after level
is completed.



<p>Finish '1' also did not appear.</p> <p>FIXED: probably due to a monster spawning on the '1' fixed by never allow monsters to spawn on '1'</p>	
<p>If save on '0' tile, quit and load again. Changes '0' to '1'.</p> <p>Fixed: characters '1' and '0' reversed when loading map.</p>	
<p>Looks like not all stars spawned – missing 1</p> <p>- may have been quitting/newgame bug fixed later on.</p> <p>FIXED: monsters may have spawned on star fixed by changing how monsters spawn into map so definitely cannot appear on same tile.</p>	

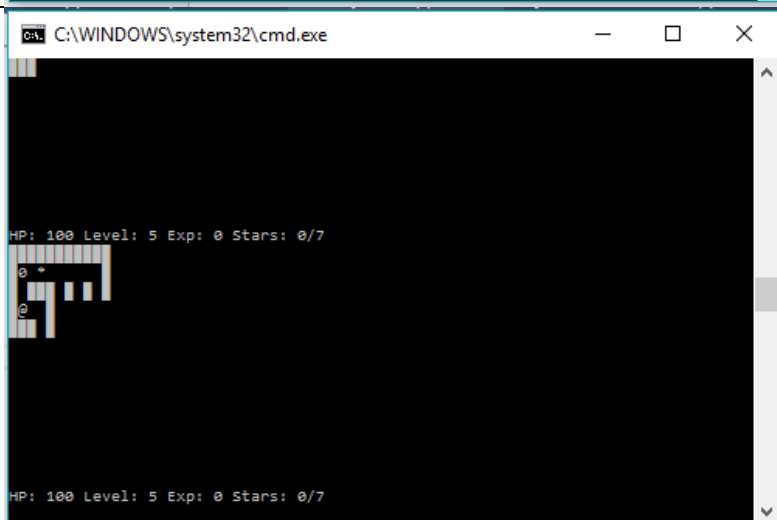
After saving and quitting then trying to exit the game, prints the map once (maybe runs game loop). Then reprints the menu again.

FIXED: Quitting the game exits it completely



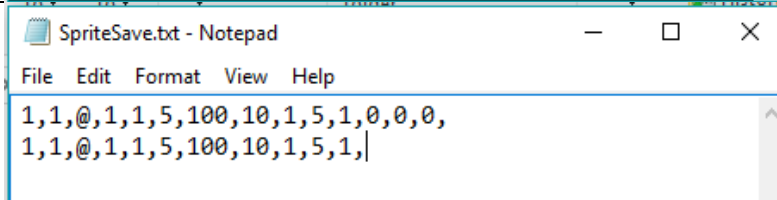
After exiting the game need to reset ALL values because star count has not reset and probably some other values.

Fixed: reset the starCount value when creating a newgame so starCount value not carried over.



Saves player values as a monster in monster array.
- only when there are no monsters on the map

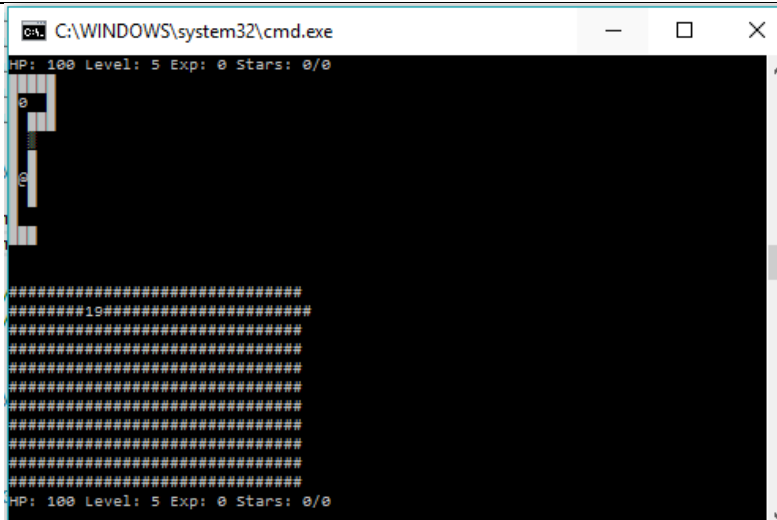
Fixed: check to see if no monsters first. If there are no monsters then skip the rest of the saving process.



Doesn't always create distance map correctly
may need to rehaul path finding for monsters.

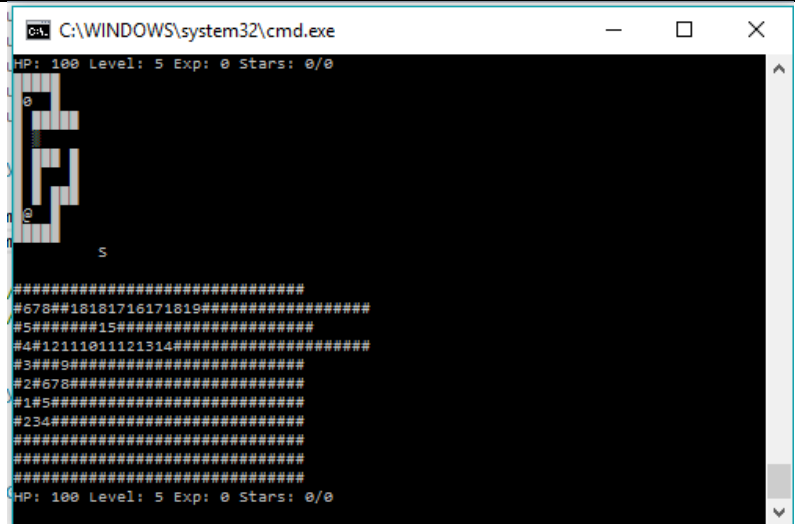
- only displays 19
- strange how not over 0 for player shows.
- must be one element left in queue?

Fixed: forgot to clear queue after each turn



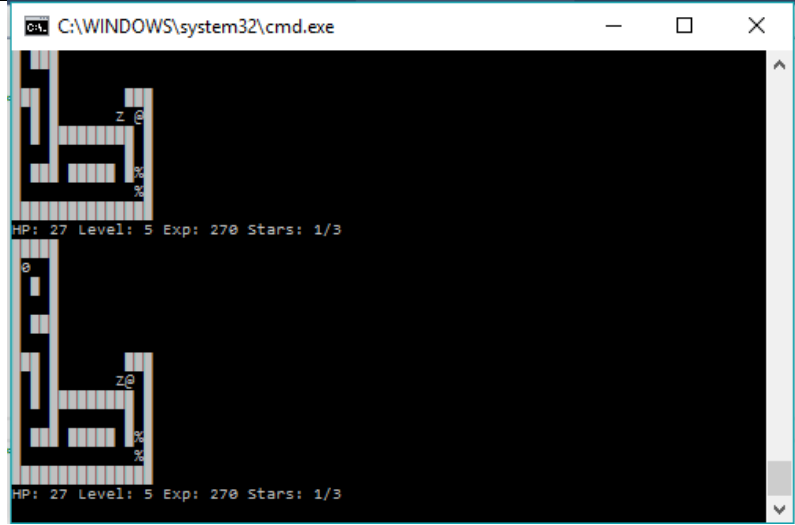
Distance map appears to lag behind player sometimes. – usually when walk into wall then move in a moveable direction.

Fixed: Removed a feature to fix and clearing queue after each turn

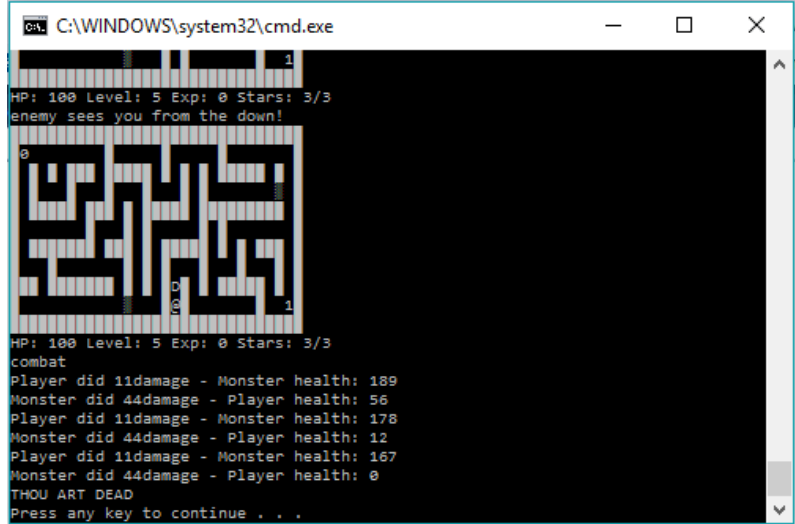


Unable to move past/fight solid enemy

Unable to identify problem however doesn't occur now.

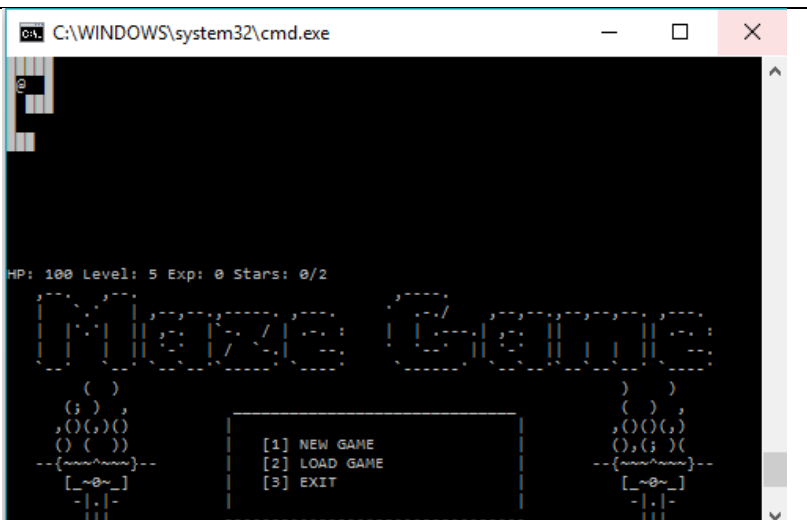


Monster and player damages not random
FIXED: only seeded srand() once at the beginning of the program



after being killed. Returns to menu screen – unable to exit – reprint screen. Press newname, create newgame but reprint menu.

FIXED: Game exits after death



Monsters cannot fight player

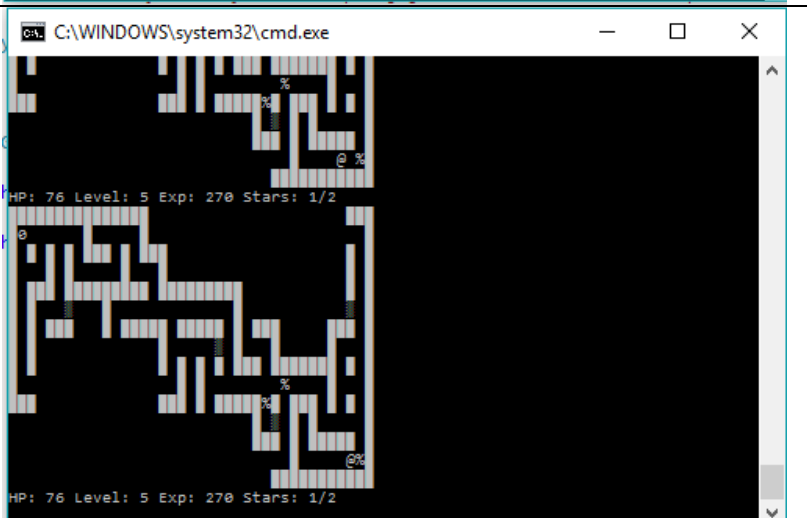
Will have to change a lot of the system to enable monsters to fight the player. Monster has no way of accessing the players information to enter combat with it.




One solution. Maybe should have added a player variable to the Game System which the monster can access. Each game system should really have a player. If created game again would design like that.



Killing a monster on an important tile still prints the '%' character. – it shouldn't

Fixed: change an if/else statement into two if's now it works



<p>too many stars in map</p> <p>Unable to identify bug however never come across it again.</p>	
<p>Monsters duplicate after loading map</p> <p>FIXED: last monster in file would be added to the monster vector twice.</p>	
<p>Monsters not showing visible when game loaded</p>	

User testing:

Three friends agreed to test the game and report any bugs and issues they discovered while playing the game. A number of reoccurring issues are stated below.

- Players said dragons spawn way to much
- Maze looked explored but not all paths had been visited making some stars very hard to find.
- Monsters spawned only in centre of map
- Some monsters too hard
- Got bored quickly

These issue are addressed in the development log and the game was changed accordingly. One issue raised was that the game become boring quickly. This issue was not addressed within the game however one solution to this problem would be to provide much more content for the game. This content could include different picks ups, armour, weapons, monsters and level design. Another method would be to provide graphics for the user to look at.

Conclusion

The complexity of the assignment came in many forms. First of all, one difficulty is to plan ahead and understand how different features will puzzle together to fit the specification described in the design document and then programming each feature together and solving any problems that arise. Many unforeseen issues occurred which had to be solved during the programming process. Another difficult aspect of the programming assignment was problem solving bugs within the program. Pin pointing the exact issue within the code could be very time consuming and stressful.

However, once complete the final product was very satisfying and the assignment provided knowledge which will be very useful in future projects. One very interesting aspect to learn about was Primm's random maze algorithm because it was a challenge to program but very fulfilling once complete. Another big part of the program was learning how to manipulate a 2D array to simulate the movement of entities in the console and how they would interact with the objects and other entities. One of the more complex problems learnt while programing was calculating how enemies would track the player using a breadth first method of searching. This also provided a practical use for working with queues. An additional feature within the program is the ability to save and load the current game to and from a file in CSV format. This information learnt can now be used in a wide range of circumstances. When looking retrospectively at the final product produced there are a number of things that should possibly be changed. One such example is handling monster and player move checks separately. A player variable would also be added to the Game system so functions can access the player's attributes easily. This change would solve the issue with monsters not being able to attack players and complexities such as passing player pointers in functions. Another feature that

could change is how spawn rates work, by giving monsters a spawn rate variable which determines their chance of spawning rather than having switch statement determining how often they spawn.

The assignment proved difficult and time consuming. However, I believe the produced work is completed to high level in accordance with the standards I set for myself. I am happy with the work produced and have learnt a lot during the process I was also able to include and complete most of the features planned out in the design document. It has also inspired me to work on improving the game itself and my skills as a programmer.