Class 10 - Group 2 PacMine

Members: Athena, Claire, Clement, Jason, Rohit

Game description

Pacmine is a modified version of the popular retro game, Pacman.

The objective of the game is to get the highest score possible through surviving for as long as possible.

The player gets to control the yellow dot, Pacman, through the arrow keys. The arrow keys control the direction in which Pacman moves. Pacman must avoid the different hazards in the maze, one being the ghost that moves around the maze and the other being the randomly generated mines. These mines are randomly spawned every 5 seconds. The player is given a 1 second reaction time where he is invincible against the newly generated mines for 1 second. The game ends when Pacman touches any of these hazards. The score acquired is based on the amount of time the Player manages to survive in the maze.

References:

We took reference from wikipedia, https://en.wikipedia.org/wiki/Maze_generation_algorithm#Recursive_implementation, for the algorithm to randomly generate the map. We also referred to the following website for the logic behind the original game of pacman: http://www.grantjenks.com/docs/freegames/pacman.html

Code documentation

This game uses the following libraries:

- Random
- Time
- Threading
- Turtle
- Colorsys

The code can be broken down into 5 different parts - interface, movement, spawning, game rules and set up.

Interface

The interface is a randomly generated maze with additional walls removed so as to increase mobility of the player. There are 4 functions used to generate the interface. This maze is generated using the coordinate system and then displayed on turtle using pixels.

gen_field(size).

1. Initialise a 2D array using a nested list of elements 0.
2. Assign every alternate element as 2 to produce a grid of ‘2’s (Fig. 1).

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Fig. 1

walk(field, start). This function uses depth recursion to generate a continuous maze.

1. start is the position of the first cell in the grid (i.e. top left integer 2 in Fig. 1)
2. Randomly choose an adjacent unvisited cell
3. Change the wall between the 2 cells to integer 1.
4. Change the value of the start cell from integer 2 to integer 1, marking as a visited cell.
5. Push the start cell to a stack of visited cells. This is to enable back tracking in step 8 later.
6. Update the start variable to the chosen cell in step 2.
7. Repeat steps 2 to 6 until the current cell has no adjacent unvisited cells.
8. Pop the cells in the stack until there is one with unvisited cells adjacent to it.
9. Continue to traverse the adjacent unvisited cells.
10. This function recurs until the stack is empty.

knock_walls(field). This function removes additional walls from the above maze.

1. Traverse the 2D array and append the coordinates (indexes) of each wall to a list of walls.
2. Randomly choose walls to be knocked off
3. Change the value of each chosen element from integer 0 to integer 1.

draw_square(t, x, y). This function draws each individual wall. Each wall is represented by a black square of length 12 pixels in turtle.

draw_map(field, t). This function traverses the field and draws black squares at the respective positions of the walls.

gen_and_draw_map(t, truestart). This function combines gen_field(size), walk(field, start) and knock_walls(map) to form a map. It then maps the existing maze in coordinate form to that drawn in turtle.

game_start(). This function creates the opening title screen.

game_over(). This function changes the state from dead = False to dead = True to activate update functions.

game_end(). This function creates the ending title screen.

Spawning

spawn_items(field, number). This function returns a list of randomly chosen positions (coordinates) on the map that is not a wall.

spawn_mines(). This function uses spawn_items(field, number) to randomly generate the position(s) of mine(s) and update the integer value of each position on the grid from 1 to integer 2. It then displays each mine at each position as red dots in turtle. This function is repeated every 5 seconds, spawning new mines in new positions every time.

Movement

Each cell in gen_field(size) is represented by a 12 by 12 pixels square in turtle. Pacman moves one pixel, named as turtle_point, with every iteration of pacman_travel(). The following functions map the turtle_points displayed on turtle to the coordinates of the cell in the grid.

coord_convert(x, y). This function converts x and y in terms of pixels to our coordinate system for the grid, returning a list [y, x].

turtle_point(coord). This function converts coordinates to pixels, accepting a list [y, x] as a parameter and returning a list [x, y].

pacman_travel(). This function states the logic behind the movement of pacman and ghost.

1. Validate the next move for both pacman and ghost
2. If not, the ghost will randomly choose an available direction.
3. And Pacman will remain at its current position

validate_<direction>_move(t). This function checks if the next move in the <direction> direction is valid.

pacman_movement(). This function listens to keyboard input to control the direction of pacman.

turning(nextheading). This function changes the heading of pacman if the next move is valid.

Game rules

countdown_timer(). This function displays how long the player has survived for. It also calculates the final score of the player when the game ends, then displays the score on the turtle screen.

fix_timer(). This function assigns countdown_timer() to its own thread such that it can run simultaneously with the other functions.

Set up

Creates the following 9 objects:

1. wn for the screen
2. firefly for the opening title screen
3. path for the map
4. pacman for Pacman
5. mines for the mines
6. title to display text
7. ghost for the ghost
8. score for the score
9. timer_title to display the time taken