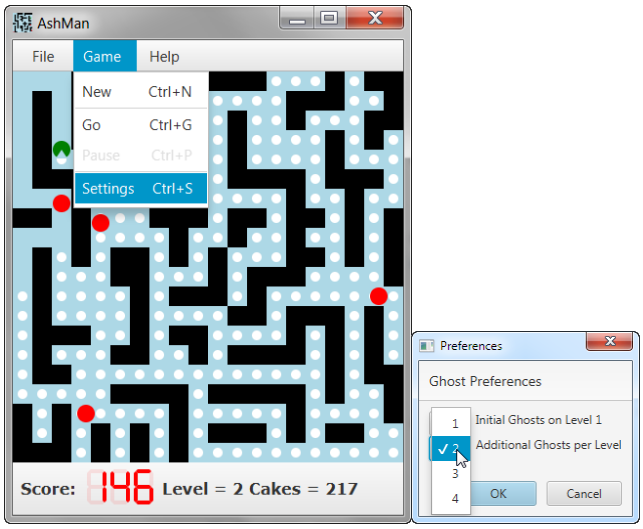
CSCD 370, GUI Programming with JavaFX
Term Project: Pac-man clone

In the early days of video games, Namco (a Japanese company) released a popular game known as Pac-man. It was first released in 1980 and was licensed in the United States by Midway (a console developer). Several successively more complex versions followed. It was also released in various other forms for home entertainment, including versions for Atari, Commodore, Nintendo, PlayStation, Xbox, PC, phones and tablets. Although dated, the game is still popular, and there are many clones for various platforms. The trademark Pac-man is owned by Bandai Namco Entertainment. In order to demonstrate your JavaFX skills, you will create a simplified "clone" of the game (mine is called AshMan, in honor of the lead character in one of my favorite B-movie series). It will look something like this:



You are NOT allowed to use an SDK other than JavaFX for this project. I'm well aware that there are sources out there for Pac-Man, including for JavaFX (at least earlier versions). I believe my requirements are sufficiently different that it won't be worth your time, but please follow the rules specified in "Cheating FAQ" if you make use of such sources. If you are in doubt about your use, please ask. If you want to make use of my sound files, just ask.

Following are specifications for the project, along with some hints on how to proceed. Read the specifications carefully, as each requirement has points associated with it. I am also providing a preliminary scoring rubric (I reserve the right to make minor adjustments). If you are uncertain about any of the requirements, please ask. I suggest you get started ASAP in order to maximize your opportunities to receive my help. A demo executable solution will be made available on my website (sorry, it will only run on Oracle Java – it will not run on Open JDK).

Overview

The player is presented with a maze. The keyboard arrow keys are used to control the movement of Ash (the green dot). Once moving, Ash keeps moving in the same direction until you change his direction or he hits a wall (in which case he stops). The maze is filled with cakes (white dots). To win the game you must eat all the cakes on two levels. There are also ghosts in the maze (red dots that move at random). If a ghost touches Ash, you lose. The first level, by default, contains two ghosts that move at half the speed of Ash. The second level, by default, adds two additional ghosts, and they move at the same speed as Ash. There are preferences that allow you to change the initial number of ghosts or the number of ghosts added in the second level.

Specifications (see the scoring rubric for a summary and point values)

1. Your game must consist of a square maze that is precisely 20x20 cells. You may make up your own maze, and are allowed to provide different mazes for the two levels and/or multiple mazes that are selected randomly. You may wrap one side of the maze to another and/or include transport portals from one place to another.
2. On startup, the main window should size itself to the game grid. The main window must be resizable and the game grid must resize with it and maintain a square aspect ratio.
3. There should be an about box containing your name, the course, and the term.

4. At the start of a game, any spot in the maze that does not contain a wall, Ash, or a ghost, should be populated by a cake.
5. The player should be able to control the movement of Ash using the arrow keys on the keyboard, just like the demo. Note that once you start Ash moving, he keeps moving in the same direction until he encounters a wall in the maze, or until you press an arrow key that changes his direction (note that you can thus stop his movement by turning him into a wall of the maze). You will receive only partial credit if the player must continually press a key to keep Ash moving. Nor should your solution cause Ash to move faster if a key is held down.
6. Ash should eat the cakes he moves over, and ghosts should not. Ghosts should either pass over each other undisturbed or change direction when they contact each other. If a ghost touches Ash, the level ends. You can detect this as them occupying the same cell. Neither Ash nor ghosts can move over walls. You may add items to your maze that give Ash special powers when he eats them, like invulnerability to ghosts for some period. If you choose to add such features, you should change Ash's appearance during the time he contains the special power.
7. Ash's mouth should move when he is moving.
8. Your program should move Ash and the ghosts smoothly between the cells of the maze, as opposed to jumping directly from cell to cell. The demo moves Ash and the ghosts between cells using four sub steps.
9. The ghosts in the demo move in one direction until they encounter a wall. At that point, they choose a new direction randomly. You may use the same approach or come up with your own algorithm to control their motion. For example, you could have your ghosts choose a direction based on Ash's location, or give a higher probability that they will move in that direction. See the random() function in the Math package.
10. The display should include a continuously updating score, the current level, and a count of cakes remaining for the current level. A convenient place for this is the status bar in our template application. Note that I made use of my seven-segment lab solution for the score. You need not do anything that fancy – it is sufficient to simply use a Label as I did for the level and number of cakes left.
11. The program should provide some visual indication of winning or losing.
12. Your game should have two levels and should default to two ghosts on the first level and four ghosts on the second level. On the first level, the ghosts should move at half the speed of Ash. On the second level, the ghosts should move at the same speed as Ash.
13. When the first level is completed, the game should pause and the player notified that the next level is about to begin. The player should be required to acknowledge the notification, by pressing a button or hitting the Enter key, before the next level begins.
14. It is EXTREMELY important that you include the following "cheat", or I will be unable to grade it completely: pressing the <Home> key should remove all cakes from that level except one.
15. Your program should include the following menu options: File: Open, Save, and Exit. Game: New, Go, Pause, Settings. You may add other game control options if you wish. Any additional control options should be given defaults such that they do not interfere with the required functionality. This means, for example, that you cannot require that some other menu choices be exercised before the game will play as specified.
16. The Game→Pause selection should be disabled when the game is paused and enabled when the game is in progress. Likewise, the Game→Go selection should be disabled when the game is in progress and enabled when the Game is paused. There are several ways to accomplish that.
17. The Game→New option should be available at any time and should reset the game in progress. It should also start the new game in the paused mode. Note that the user might select this repeatedly in an effort to obtain an advantageous starting position for Ash relative to the ghosts.
18. The player should be allowed to save and load a game to or from a file, and it should not be possible to do either unless the game is paused. My Save option always prompts for a filename, which is why I called it "Save As" on the menu. Your file chooser dialogs should use an appropriate filter (mine use .ash for the file extension).
19. Your game should make use of at least two sound clips during game play. My demo has clips for eating cakes, advancing to the next level, losing, and winning.
20. You should implement at least two preferences to control the number of initial ghosts (1-4) and the number of additional ghosts on level 2 (1-4). The preferences should be persistent between runs of the program. Changing the ghost settings should not restart the game, but may re-seed all ghosts if the preferences are changed in a way that affects the current level (i.e., any preference change while in level 2, and a change to the initial ghost count if in level 1).

Advice

1. Start simple, add and test functionality incrementally. Here is a rough suggestion for a possible development sequence: Start by getting the maze initialized and drawn. Add code that populates and draws the cakes. Add code to draw Ash and the ghosts. Add an AnimationTimer and get the ghosts moving in response to the timer. Do the movement in cell-to-cell jumps at first. Add code to draw and move Ash in response to keyboard and timer events. Add menus and implement handlers for the Pause and Go menu selections. Implement collision detection. Implement the status display. Add the second level of play along with the cheat code. Add smooth movement between cells. Animate Ash's mouth. Add code to enable and disable the menu items. Add File I/O. Add Preferences. At some point when you need a break from strenuous thinking, add the about box. Test your code incrementally and backup your work regularly.
2. I implemented my maze with a simple two-dimensional array, where each cell contains one of three values describing the contents (blank, wall, or cake). This array is used to draw the maze. Ash and the ghosts are drawn on top of the maze.
3. You can add smooth movement between cells by including a "substep" variable for each ghost and Ash, which tells how far into the next cell, in the current direction, the ghost or Ash should be drawn. My substep variable counts through 4 states, with a (circular) increment on every timer hit. In my implementation the Ghosts and Ash extend a common superclass that encapsulates the direction of movement and other state information, along with methods that move the object one substep, serialize and deserialize the object to or from a stream, and an abstract method for drawing. Derived classes for Ash and the ghosts provide an implementation for the draw method.
4. Ash's mouth can be animated by drawing over a portion of his body with a triangle in the background color during the second half of his smooth movement into the next cell.
5. The only events handled directly by my Game class are timer events.

Scoring Rubric

Pts	Val	Description of requirement
	3	Main window displays a 20x20 maze with corridors, walls, and a square aspect ratio.
	3	Main window is resizable. Maze resizes with it but maintains a square aspect ratio.
	2	There is an appropriate About box
	4	Maze initializes with cakes in free spaces
	4	Keyboard arrow keys move Ash
	4	Ash keeps moving until encountering a wall or his direction is changed
	3	Holding a key down does NOT speed Ash up
	4	Ash eats cakes, ghosts do not
	4	Ash and ghosts move smoothly between cells
	3	Ash's mouth moves smoothly
	4	Ghosts move continuously
	4	Ghosts change direction on encountering a wall
	3	Ghosts do not get stuck on the same path
	4	Score, level, and number of cakes left is displayed and updated
	4	Ghost collision with Ash ends the current level
	2	Level change pauses with a notification that must be dismissed by the player before continuing
	2	End of game gives different indications for winning and losing
	4	There is a working second level with more and faster ghosts
	4	Working "cheat key" (Home) removes all but one cake from the current level
	6	Working New, Pause, and Go options
	2	Correct enabling and disabling of Go and Pause
	6	Working Save and Open
	2	Open and Save are disabled unless the game is paused
	4	At least two sound clips are used during game play
	2	Working menu shortcuts and accelerators
	6	Functioning preferences for initial ghosts and ghosts to add
	3	Preferences are persistent between runs
	2	Changing ghost preferences retains the game state except perhaps for ghost placement
	2	Changing number of added ghosts does not change any game state if currently in level 1
	100	Total Score