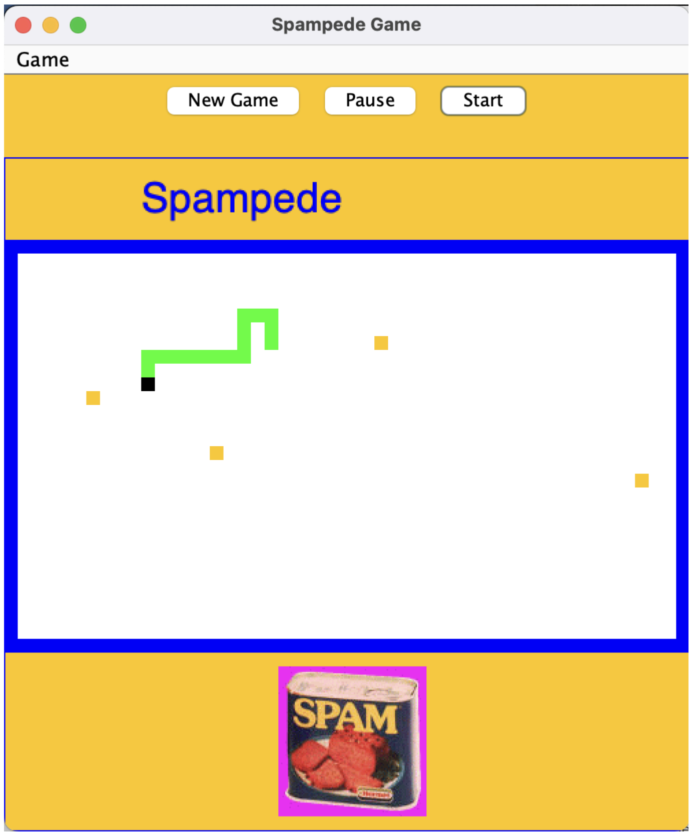
HW10: Spampede!

The learning objectives of this homework are:

- Become familiar with the Model-View-Controller pattern of object-oriented programming for programs with a graphical user interface
- Gain practice working with an established codebase
- Build a fun game!

The Spampede game is a variation of the familiar Snake game in which you maneuver a growing snake around a board to eat dots of spam. Press Start to begin the first game, or New Game to restart. The game is controlled by the *ijkl* keys to move up, left, down, or right, respectively. Pressing *a* enables an AI mode in which the snake hunts spam autonomously.



The game is implemented with three interacting classes and a few helper classes:
- SpampedeModel: store the game board and handle game logic
- SpampedeView: draw the board and user interface in a window
- SpampedeController: main method, respond to user clicks and keypresses

Note: This assignment is two weeks long and counts as two-assignments' worth of points. It is substantially longer than previous assignments. Work on it gradually rather than trying to do it all right before the deadline.

As always, please let us know if you have questions or get stuck -- we are happy to help!

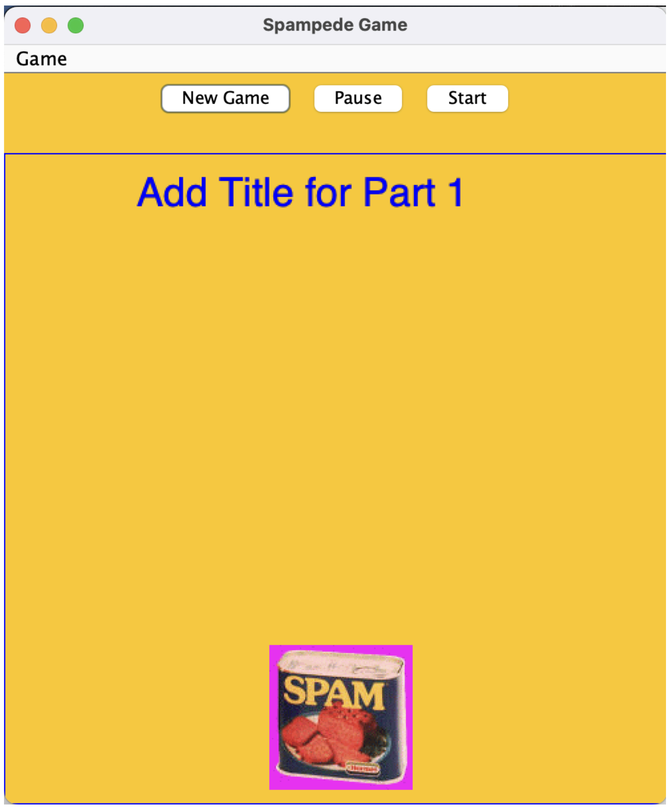
— Your profs & your awesome CS60 Grutoring team!

Part 0: Getting Started

Pull hw10 from the CS60 Github repository. Create a new hw10 VSCode project. Add all the .java files to hw10/src/com/gradescope/spampede. Add the crunch.wav, Spam.au, and spam.gif to the top level (hw10) directory.

Go to the flask icon, enable Java Testing (JUnit), and run all the tests. BoardCellTest should pass and all the others should fail, with many problems reported such as methods not returning results and methods unused.

Run the SpampedeController.java main function. You should see the following window appear. Remember to close the window after each time you run Spampede, so you don't have many copies cluttering up your system.

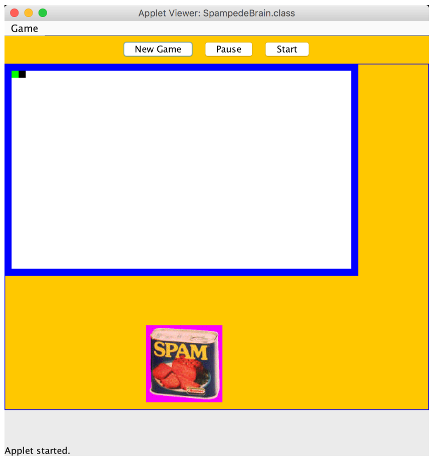


Part 1: Draw the board

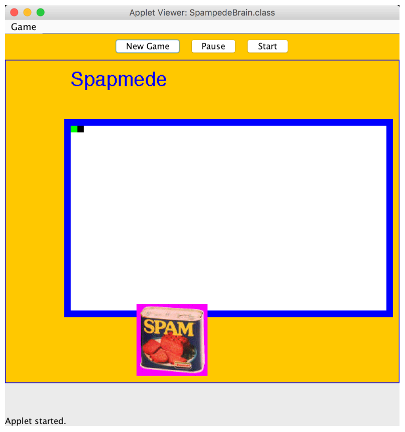
Task: Draw the board (which contains the walls, the snake, the spam, and the empty cells), and fix the typo in the title of our game.
Code: `SpampedeView.java: updateGraphics()`, `Preferences.java`

- **Fix the title of the game in `Preferences.java`** to be "Spampede."

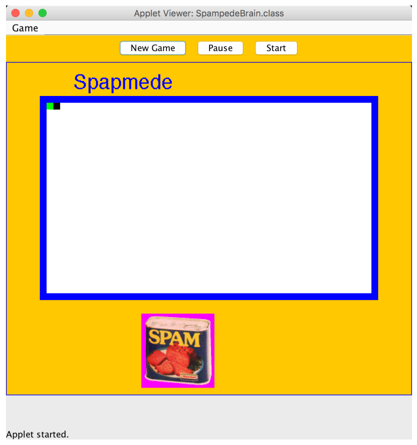
- **Modify `updateGraphics()` in `SpampedeView.java`** to draw the board based on the contents of the model (`model`). Because a lot of useful functionality is already present in other methods, you should not have to write very much code (10 lines or less)
  - *Hint*: Be careful to correctly match `row` vs `column` and `x` vs `y`. Note that a cell is larger than one pixel (see `Preferences.CELL_SIZE`).
  - *Hint*: Look at `getCellColor` in `SpampedeModel.java`.
  - After you complete this step, you should see the following board:



- **Shift the board**
  - Modify the code in `updateGraphics()` to **shift the board 90 pixels down and to the right**. You should see the title and the following board:



  - Oops, 90 pixels was not quite right – the board is too far down and to the right. While we could play around with other numbers, **add code that *computes* a good offset so that the board is centered horizontally and looks nice vertically**. You should use `this.width`, and you can use constants from the `Preferences` class (e.g. `Preferences.CONSTANT_NAME`).



- Once you have completed this part, please **take a screenshot to submit later as `Spampede.jpg`**.
- Submit the following files to the Gradescope assignment "HW #10.1 Spampede board":
  - `Spampede.jpg`
  - `SpampedeView.java`
  - `Preferences.java`
- Note about submitting image files:
  - Sometimes your computer will hide the file extension (e.g. .jpg), which can make it harder to submit a file with the right name!
  - View file extensions on Windows 10 or Mac
  - Convert to jpg: Google can direct you to some useful resources

Note: For the remaining parts, you can submit your files (`SpampedeModel.java` and `SpampedeController.java`) to the Gradescope assignment "HW #10.2-6 Spampede functionality".

Part 2: Get neighboring cells

Task: Write code that can determine the location of a snake's neighbors on all sides. We will also write code that can determine to which neighbor the snake will move next (based on its current location and the direction in which it is heading).

Code: `SpampedeModel.java`: `get...Neighbor(BoardCell)` and `getNextCellInDir()`

- **Implement the four `get...Neighbor(BoardCell)` methods in `SpampedeModel.java`** so that the tests in `SpampedeModelTest_Neighbors.java` pass.
  - *Note*: You should never make new `BoardCells`! Instead, return a reference to `BoardCells` that are already within the `SpampedeModel` object.
- **Implement `getNextCellInDir()` in `SpampedeModel.java`** so that the tests in `SpampedeModelTest_CellInDir.java` pass.
  - *Hint*: Use `this.currentMode` and provided methods.
  - *Optional*: You are welcome to use a `switch` statement rather `if-then-else`.
- You might find it helpful to look at pictures of the test boards.

Part 3: Move the snake within the board

Task: Write code that causes the snake to move around the board.

Code: `SpampedeModel.java`: `moveSnakeForward(BoardCell)`, and possibly helper methods

- **Implement** `moveSnakeForward(BoardCell)`, which moves the snake to the cell provided as an argument. Keep in mind the following.
  - Requirements:
    - You might want to add a method (or two) in `SpampedeModel.java` to modify the data for your game (e.g. `snakeCells`) as desired. **Add these helper methods in the section titled "`Snake movement methods`".**
    - You should not create any new `BoardCells` or modify the row or column of any existing `BoardCells`. Instead, you should use the `BoardCell` methods `becomeHead()`, `becomeBody()`, and `becomeOpen()`. That is, delegate when possible (i.e. rely on methods and hide implementation details).
  - Implementation details:

        - `snakeCells` is of type `LinkedList<BoardCell>`, which is the Java-library version of a *doubly-linked list* (a list in which each node has references to the previous and next nodes). Check out its documentation.
        - If `nextCell` is not spam, the snake should move. That is, `nextCell` will be the new head, and the last piece of the tail will no longer be part of the snake.
        - If `nextCell` is spam, the head should be moved forward (and the tail stays the same spot).
- After completing this part, the tests in `SpampedeModelTest_UpdateSnake.java` should pass.

Part 4: Handle (some) key presses

Task: Allow the player to move the snake, by pressing keys on the keyboard
Code: `SpampedeController.java`: `keyPressed(KeyEvent)`

- **Add the missing cases to `keyPressed(KeyEvent)` in `SpampedeController.java`** so that when you press the relevant key, it updates the snake direction.
    - *Hint*: Use the `setDirection...()` methods in `SpampedeModel.java`.
    - No magic strings (or characters) please! Use the provided constants for key mappings (e.g. follow the example that uses `PLAY_SPAM_NOISE`).
- While you can write this code using `if-then-else` statements, you are probably better off using `switch` statements.
    - *Note*: By default, Java "falls through" switch statements, as in all statements after the matching case label are executed in sequence. To prevent fall-through, put the keyword `break` at the end of each case!

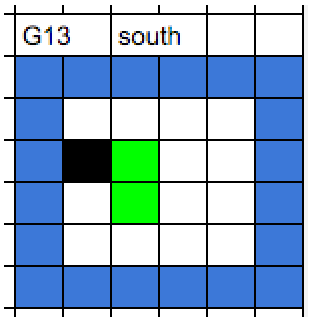
After completing this part, you should be able to run `SpampedeController.java` and play the Spampede game!

Note: At this point we have covered 85% of the grade for this assignment (including style). You can stop here if you want, or go on to implement reversal and AI for the remaining 15% of the grade.

Part 5: Write code to reverse the snake

Task: Make it possible for the snake to reverse itself.
Code: `SpampedeModel.java`: `reverseSnake()`

- **Implement `reverseSnake()` in `SpampedeModel.java`**, which reverses the snake.
    - *Hint*: Look at pictures of the test boards, and try to manually compute the new direction of the snake. The new direction does **not** involve the old direction of the snake, only the position of the new head and new neck. For example, when the snake below is reversed, it will be going South.



- After completing this part, the tests in `SpampedeModelTest_Reverse.java` should pass. In addition, you should be able to press the `'r'` key during the game and have the snake reverse.
    - *Note*: These (and all) test cases are not guaranteed to be comprehensive. It is possible that your code will have errors that are not caught by these test cases, and we encourage you to test the functionality within the game to help catch additional errors.

Part 6: Search for Spam: Implement AI Mode!

Task: Implement a search-based AI that can automatically play the game
Code: `SpampedeModel.java`: `getNextCellFromBFS()` and helper method `getFirstCellInPath(BoardCell)`

- **Complete `getNextCellFromBFS()` in `SpampedeModel.java`**, which returns the `BoardCell` on the path to the nearest spam.
- In writing BFS, you will need to decide the order in which you add the neighbors of each cell to the Queue.
    - To satisfy our test cases, you should add the neighbors in the following order: North, South, East, West. (*Hint*: This order is already encoded in a provided method of **`SpampedeModel.java`**. Find the method (and use it)!
    - So, if there are two spams that are equidistant, you should return the path to the one you found first (i.e. based upon adding the neighbors to the queue in the above order).
- Read the test cases. After completing this part, you should pass the tests in these files:

- `SpampedeModelTest_CheckParentsBFS.java`
    - This test file uses a method of **white-box testing**, where we look inside the structures (here, we look at the parent references of `BoardCells`) to make sure that the algorithm is behaving as intended.
    - These tests assume the BFS algorithm stops when a target (spam) is *dequeued*.
- `SpampedeModelTest_GetNextCellFromBFS.java`
    - This test file uses a method of **black-box testing**, where we just test the functionality of the methods without needing to know the algorithm or data structures used.
- In addition, you should be able to press the `'a'` key during the game and have the snake find spam automatically!

Part 7: Submit!

- As always, please check your code against the CS 60 style guide.
- Submit the following files:
    - To the Gradescope assignment "HW #10.1 Spampede board"
        - `Spampede.jpg`
        - `SpampedeView.java`
        - `Preferences.java`
    - To the Gradescope assignment "HW #10.2-6 Spampede functionality"
        - `SpampedeModel.java`
        - `SpampedeController.java`

Rubric

#	Name	Autograder	Functionality	Style	Total
1	Draw the board	0	12	30	12
2	Neighbors	24	0		24
3	Move the snake	22	8		30
4	Key presses	0	6		6
5	Reverse	4	2		6
6	AI	11	1		12
		50.83%	24.17%	25.00%	120