Snake Clone (Game Design Document)

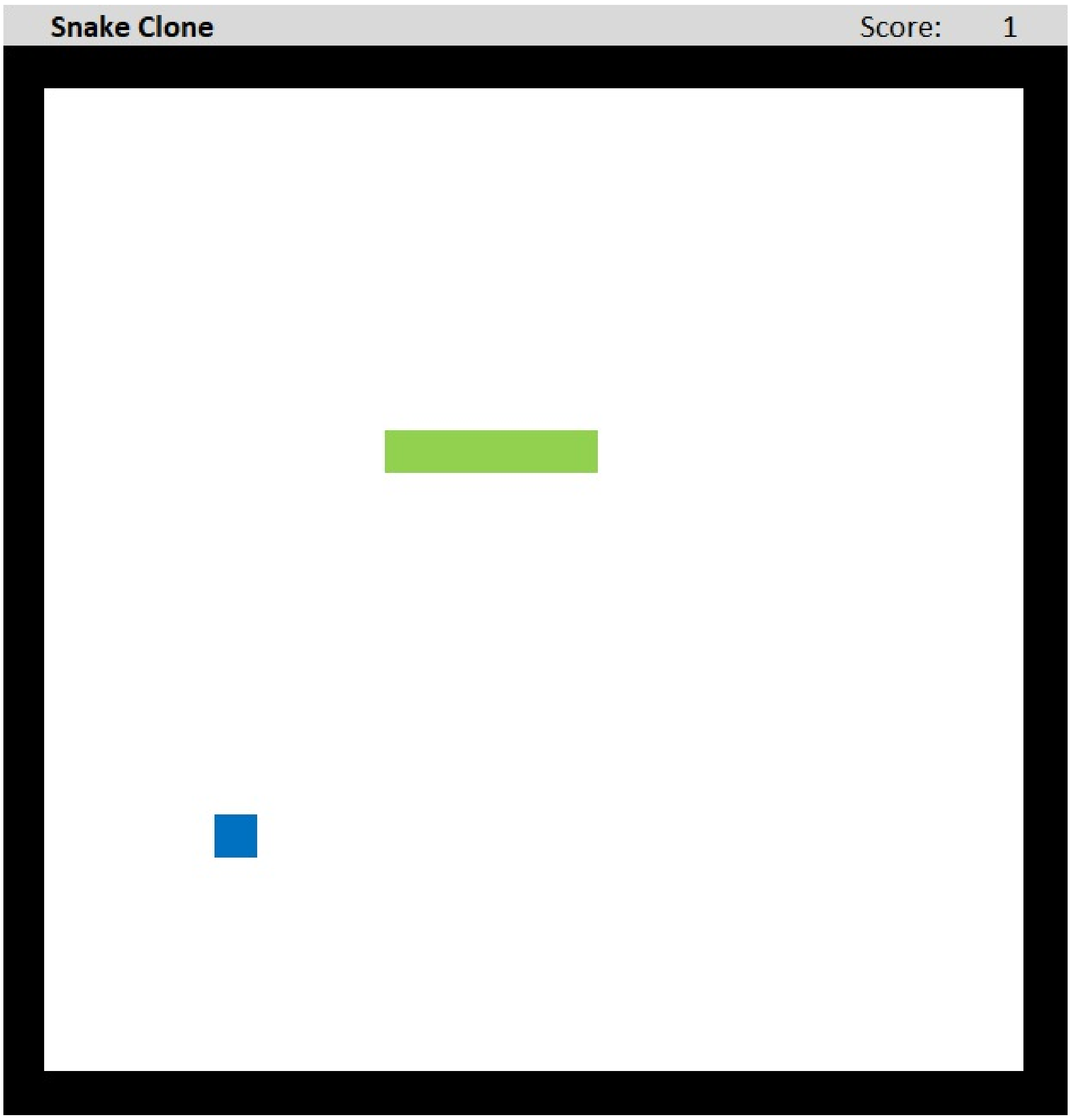
Introduction

Snake is a video game that w as first implemented in the mid 1970s in arcades. The objective of the Snake Clone project is to implement a clone of this classic game using JavaScript. More details about Snake can be found on Wikipedia.

Gameplay

- The game takes place on a 25 x 25 square grid w ith the first and last row s and columns of the grid being filled by w all tiles.
- The player controls a snake that takes up tiles based off of how much food it has eaten.
- The snake moves in the direction that its head is facing. The player can change this direction using the arrow keys.
- Food is randomly generated on a tile that the snake is not occupying. A maximum of one food item is on the board at any given time.
- If the snake's head comes into contact w ith the food tile. The food is eaten and the snake's size increases by one tile.
- If the snake's head comes into contact w ith a w all tile or w ith a tile that corresponds to the snake's ow n body, then the game is over.
- The player begins the game w ith the snake generated in a consistent location. At the start of the game, the snake takes up five tiles.
- A score is displayed in an area that is separate from the grid. The player begins w ith a score of 1. Each time the snake eats a food tile, the player gets 1 point.
- The game has three states: a title state (w here a title screen is show n and the user is prompted to start the game), a play state (w here the game is played), and a game over state (w here the game is stopped, the user can see their score and final game position, and the user can restart the game).

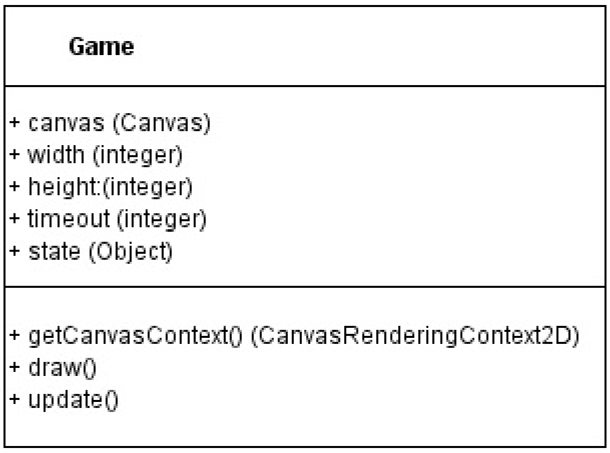
Mockup



Engine (Reusable) Classes

Game

In general, a game's main loop is modeled using a general reusable **Game** class. In the main program we iterate in a loop and call the **update()** method followed immediately by the **draw()** method until the game is finished. The **Game** class has a state attribute which is used to model the current situation that the game is in. The state object itself is expected to have its own **draw()** and **update()** operations.



Attributes

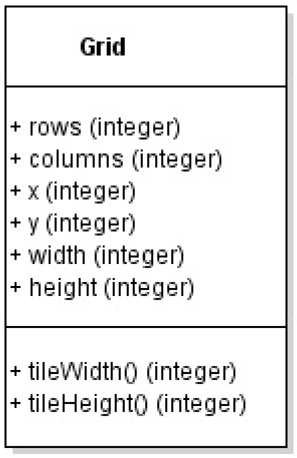
- **canvas (Canvas):** the HTML5 canvas upon which the game is drawn.
- **width (integer):** the desired width of the canvas.
- **height (integer):** the desired height of the canvas.
- **timeout (integer):** how long the main program should wait between loop iterations.
- **state (Object):** represents whether the game is currently in the title, play, or game over state.

Methods

- **getCanvasContext() (CanvasRenderingContext2D):** extracts the drawing context from the canvas (for use in both the **draw()** and **update()** operations).
- **draw() (void):** extracts the canvas context and passes it to the state's **draw()** operation.
- **update() (void):** adjusts the game canvas width and height to match the game's desired width and height and then calls the state's **update** operation.

Grid

The game grid is modeled using a general reusable **Grid** class. In the game grid there is a 25 x 25 area for the snake; food; and walls, and there is an additional row on top for the score area. The purpose of this class is to store the grid and tile dimensions for use by grid elements.



Attributes

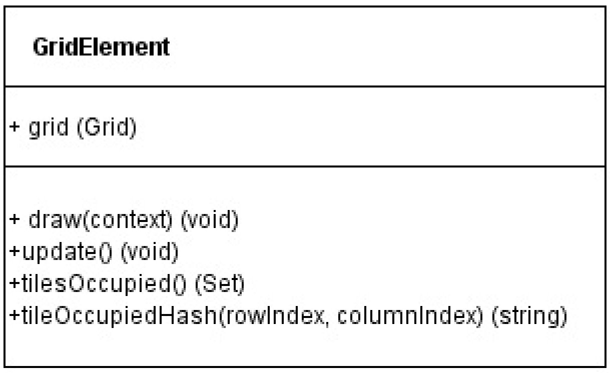
- **rows (integer):** the number of row s there are in the grid.
- **columns (integer):** the number of columns there are in the grid.
- **x (integer)**: the x coordinate that the grid has w ith respect to the canvas area.
- **y (integer)**: the y coordinate that the grid has w ith respect to the canvas area.
- **width (integer)**: the w idth that the grid has w ith respect to the canvas area.
- **height (integer)**: the height that the grid has w ith respect to the canvas area.

Methods

- **tileWidth() (integer):** the w idth of each tile in the grid; computed as **width // columns**.
- **tileHeight() (integer):** the height of each tile in the grid; computed as **height // rows**.

GridElement

To model the parts of the game that exist in the grid, a general reusable **GridElement** class is used to specify the interface that all draw n components of the game must satisfy.



Attributes

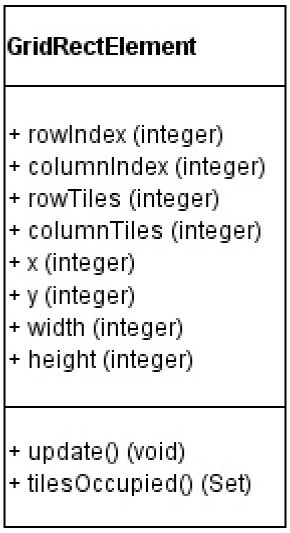
- **grid (Grid):** the grid to use w hen updating the element.

Methods

- **draw(context) (void):** displays the element on the canvas using the passed in canvas context (**CanvasRenderingContext2D**).
- **update() (void):** called at each game tick; updates the element so that the next call to draw() displays the element accurately.
- **tilesOccupied() (Set):** used for collision detection; a set of all tiles that the element is considered to be occupying.
- **tileOccupiedHash(rowIndex, columnIndex) (string):** converts a given tile's rowIndex (**integer**) and columnIndex (**integer**) into a string value for insertion and comparison within the tilesOccupied() Set.

GridRectElement

There are six kinds of **GridElements** on the grid that are rectangular in shape. These include: the game walls, game background (also known as the floor), score background, food tiles, snake tiles, and text (while text is not drawn as a rectangle, it does take up a rectangular area). A general reusable class called **GridRectElement** is used for implementing the **update()** and **tilesOccupied()** methods of these elements. The **draw()** method is left for a subclass to implement.



Attributes

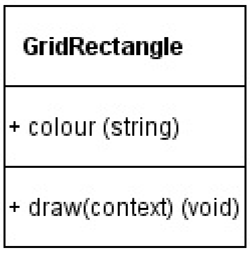
- **rowIndex (integer):** the number corresponding to the grid row of the top left corner of the element (0-based indexing is used).
- **columnIndex (integer):** the number corresponding to the grid column of the top left corner of the element (0-based indexing is used).
- **rowTiles (integer):** the number of rows that the element occupies.
- **columnTiles (integer):** the number of columns that the element occupies.
- **x (integer)**: the x coordinate that the element's top left corner has with respect to the canvas area.
- **y (integer)**: the y coordinate that the element's top left corner has with respect to the canvas area.
- **width (integer)**: the pixel width that the element has with respect to the canvas area.
- **height (integer)**: the pixel height that the element has with respect to the canvas area.

Methods

- **update() (void):** called at each game tick; changes the values of **x**, **y**, **width**, and **height** in accordance with the current values of **rowIndex**, **columnIndex**, **rowTiles**, **columnTiles**, and **grid**.
- **tilesOccupied() (Set):** returns a Set of strings corresponding of all tiles in the rectangular area between (rowIndex, columnIndex) and (rowIndex + rowTiles - 1, columnIndex + columnTiles - 1) inclusive.

GridRectangle

The game walls, game background (floor), score background, food tiles, and snake tiles are all coloured rectangles. They are thus implemented by using a **GridRectangle** class that subclasses the above **GridRectElement** class. The **GridRectangle** class has an additional attribute for the colour of the rectangle and implements the **draw()** method.



Attributes

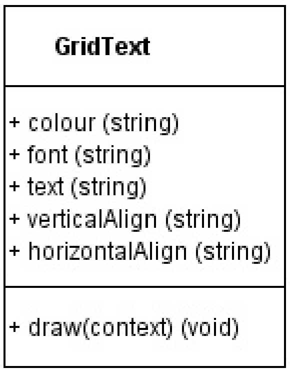
- **colour (string):** A CSS string corresponding to the colour of the rectangle.

Methods

- **draw(context) (void):** uses the fillRect method of the passed in canvas context (**CanvasRenderingContext2D**) to draw a coloured rectangle betw een (x, y) and (x + w idth, y + height).

GridText

The score area displays text including text for the title of the game as w ell as text for the score. This text is modeled using a **GridText** class that subclasses the **GridRectElement** class in order to draw the text w ithin a bounding rectangle of tiles in the grid. The class also includes attributes for vertical and horizontal alignment.



Attributes

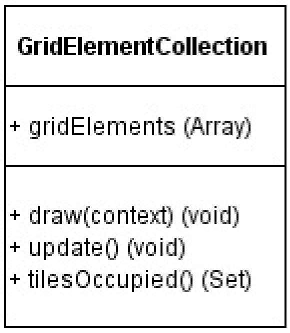
- **colour (string):** A CSS string corresponding to the colour of the text.
- **font (string):** A CSS string corresponding to the font of the text.
- **text (string):** The w ords to display.
- **verticalAlign (string):** How the text should be vertically aligned w ithin its bounding rectangle (one of "top", "middle", or "bottom").
- **horizontalAlign (string):** How the text should be horizontally aligned w ithin its bounding rectangle (one of "left", "centre", or "right").

Methods

- **draw(context) (void):** uses the fillText method of the passed in canvas context (**CanvasRenderingContext2D**) to draw the desired text betw een (x, y) and (x + w idth, y + height).

GridElementCollection

The **GridElementCollection** class is a general reusable class that is used for encapsulating several **GridElements** at once. It can draw its elements all at once and update them all at once. It can also take a union of the Sets returned by each element's tilesOccupied() method. Since the **GridElementCollection** class is able to implement its own **draw()**, **update()**, and **tilesOccupied()** methods, it subclasses the **GridElement** class. (Note that this technique is formally known as the Composite Pattern)



Attributes

- **gridElements (Array):** An array of the **GridElements** associated with the collection in the order that they should be drawn/updated.

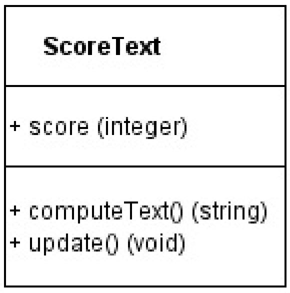
Methods

- **draw(context):** loops through gridElements, calling each one's **draw()** operation with the passed in canvas context (**CanvasRenderingContext2D**)
- **update():** loops through gridElements, calling each one's **update()** operation.
- **tilesOccupied()**: loops through gridElements, unioning all of their **tilesOccupied()** Sets together and returning the result.

Implementation-Specific Classes

ScoreText

The **ScoreText** class is an implementation-specific class because its approach for computing the text to display is not as reusable as desired. The **ScoreText** class subclasses the **GridText** class and has an additional attribute for storing the game score. The **ScoreText** class updates its text based off of what the score is and ensures that it is a right-aligned 3 character string.



Attributes

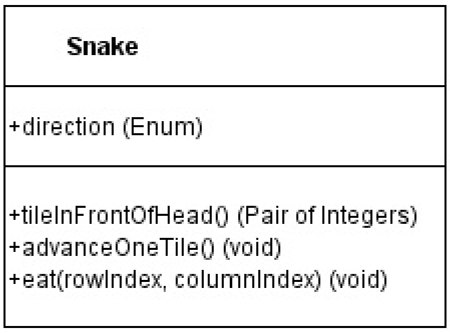
- **score (integer):** the score in the current game (a number between 0 and 999).

Methods

- **computeText() (string):** converts the game score into a right-aligned 3 character string and then returns the string for use by the **update()** operation.
- **update() (void):** Sets the text equal to what is returned by the **computeText()** operation and then calls the super class's **update()** operation.

Snake

The **Snake** class is an implementation-specific class because its use case is specific to the game of Snake. The **Snake** class subclasses the **GridElementCollection** class and stores its tiles in the super class's gridElements array. The **Snake** class also has a direction attribute for moving its head and body, as well as operations that facilitate coding the game.


Snake
+direction (Enum)
+tileInFrontOfHead() (Pair of Integers)
+advanceOneTile() (void)
+eat(rowIndex, columnIndex) (void)

Attributes

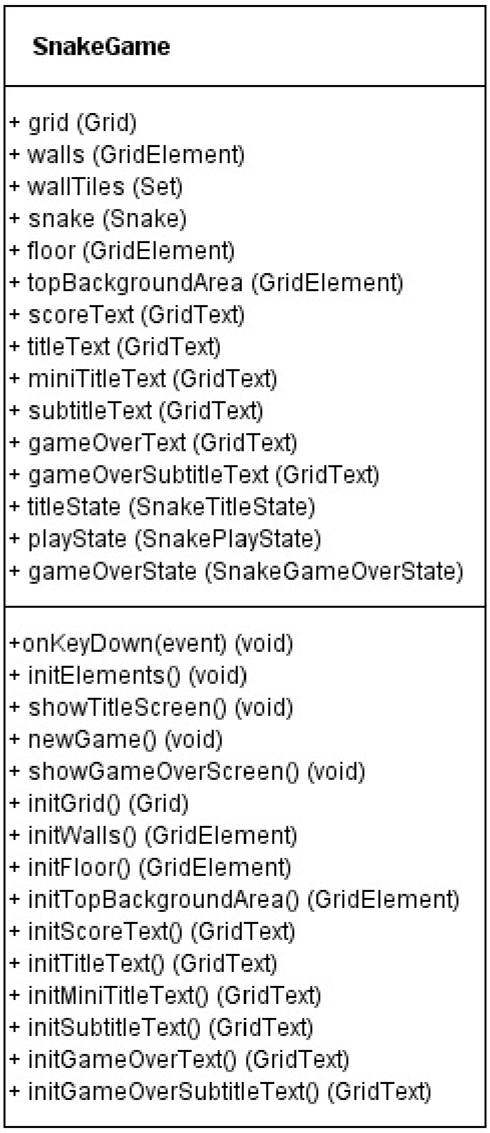
- **direction (Enum):** the direction that the snake is moving in. One of "up", "down", "left", "right".

Methods

- **tileInFrontOfHead() (Pair of Integers):** Computes and returns the row index and column index of the tile directly in front of the snake's head using the direction attribute. Used for collision detection.
- **advanceOneTile() (void):** Adjusts the Snake's tiles so that the head is moved one tile in the snake's direction and the snake's body moves along with the head.
- **eat(rowIndex, columnIndex) (void):** Adds a tile at the specified row index and column index to the snake's body.

SnakeGame

The **SnakeGame** class is an implementation-specific class because it is the implementation! This class subclasses the **Game** class and includes the code for initializing the game elements, storing the game state, operations for changing the game state, and additional overhead for handling user input.



SnakeGame

+ grid (Grid)
+ walls (GridElement)
+ wallTiles (Set)
+ snake (Snake)
+ floor (GridElement)
+ topBackgroundArea (GridElement)
+ scoreText (GridText)
+ titleText (GridText)
+ miniTitleText (GridText)
+ subtitleText (GridText)
+ gameOverText (GridText)
+ gameOverSubtitleText (GridText)
+ titleState (SnakeTitleState)
+ playState (SnakePlayState)
+ gameOverState (SnakeGameOverState)

+onKeyDown(event) (void)
+ initElements() (void)
+ showTitleScreen() (void)
+ newGame() (void)
+ showGameOverScreen() (void)
+ initGrid() (Grid)
+ initWalls() (GridElement)
+ initFloor() (GridElement)
+ initTopBackgroundArea() (GridElement)
+ initScoreText() (GridText)
+ initTitleText() (GridText)
+ initMiniTitleText() (GridText)
+ initSubtitleText() (GridText)
+ initGameOverText() (GridText)
+ initGameOverSubtitleText() (GridText)

Attributes

- **grid (Grid):** The **Grid** instance used by the game.
- **walls (GridElement):** The **GridElement** that represents the boundary of the game. Should the snake leave the boundary. It is game over!
- **wallTiles (Set):** A **Set** that corresponds to the **tilesOccupied()** of the **walls** attribute. Note that **wallTiles** is computed only once (when we call **initElements**) so that we don't have to keep calling **tilesOccupied()**.
- **snake (Snake):** The **Snake** that corresponds with game element that the player moves with the arrow keys.
- **floor (GridElement):** The **GridElement** used for drawing the background area upon which the snake is allowed to move.
- **topBackgroundArea (GridElement):** The **GridElement** used for drawing the background area upon which the score text is displayed.
- **scoreText (GridText):** The **GridText** that displays the game score.
- **titleText (GridText):** The **GridText** that displays the main title on the title screen.
- **miniTitleText (GridText):** The **GridText** that displays the game title in the score area.
- **subtitleText (GridText):** The **GridText** that displays text below the main title on the title screen.
- **gameOverText (GridText):** The **GridText** that displays the text "Game Over" in place of the game title in the score area when the game is over.
- **gameOverSubtitleText (GridText):** The **GridText** that displays additional instructions (regarding how to start a new game) when the game is over.
- **titleState (SnakeTitleState):** The game state that corresponds to displaying the title screen.
- **playState (SnakePlayState):** The game state that corresponds to playing the game.
- **gameOverState (SnakeGameOverState):** The game state that corresponds to the game being over.

Methods

- **onKeyDown(event) (void):** Handles the overhead involved with user keyboard input and passes the event to the current game state.
- **initElements() (void):** Calls all of the over **init** operations to initialize the game elements and then initializes the game state objects. Also computes the value for the **wallTiles** attribute.
- **showTitleScreen() (void):** Changes the game state to **titleState**.
- **newGame() (void):** Changes the game state to **playState** and starts a new game.
- **showGameOverScreen() (void):** Changes the game state to **gameOverState**.
- **initGrid() (Grid):** Returns an initialized value for the **grid** attribute.
- **initWalls() (GridElement):** Returns an initialized value for the **walls** attribute.
- **initFloor() (GridElement):** Returns an initialized value for the **floor** attribute.
- **initTopBackgroundArea() (GridElement):** Returns an initialized value for the **topBackgroundArea** attribute.
- **initScoreText() (GridText):** Returns an initialized value for the **scoreText** attribute.
- **initTitleText() (GridText):** Returns an initialized value for the **titleText** attribute.
- **initMiniTitleText() (GridText):** Returns an initialized value for the **miniTitleText** attribute.
- **initSubtitleText() (GridText):** Returns an initialized value for the **subtitleText** attribute.
- **initGameOverText() (GridText):** Returns an initialized value for the **gameOverText** attribute.
- **initGameOverSubtitleText() (GridText):** Returns an initialized value for the **gameOverSubtitleText** attribute.

Outstanding Issues

- Game Design Document needs a navigation zone to make jumping to sections and subsections easier.
- Game state classes to be refactored and documented.
- Class diagram showing all classes and the inheritance structure to be added.
- Support for dynamic sizing of the game screen to be added.
- Graphics to be enhanced with images instead of raw colours.
- An eating animation to be added for when the snake eats a tile.
- Sound effects to be added.
- High scores to be added.