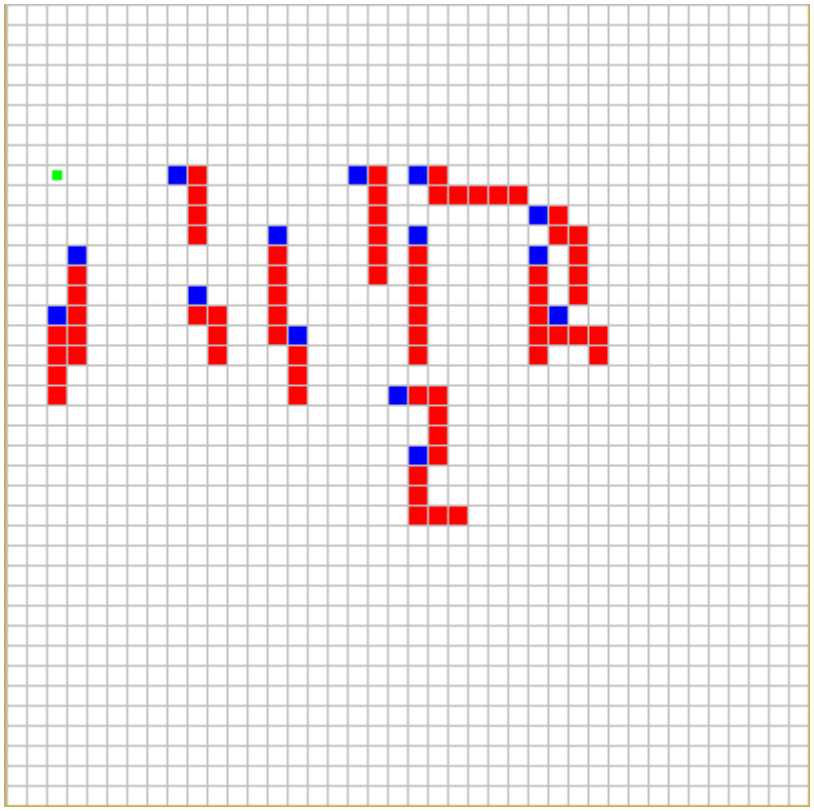
CmpE 160 Introduction to Object Oriented Programming, Spring 2017 – Project 2

Project Report Second Project



Halil Umut Özdemir | Snakes! | 28.04.2018

1. Description:

For this project we implemented a game called Snakes. This game is like the classical snake game but there are some differences and improvements. Like classical game, game starts with 1 snake but in this game user cannot control the snake. Snake will move with using its simple AI. Snake finds to food and eats it then it grows 1 square. When the size of snake reaches 8, snake will reproduce and generates a new snake. This is the simple explanation of the game.

2. Class Hierarchy:

The *ui* package includes the classes for creating the user interface. With the use of this package game creates a visuable grid for my snakes and food.

The *snakesimulator* package includes 2 classes GridGame and SnakeSimulator. GridGame creates a printable panel our game and stores the set of drawable

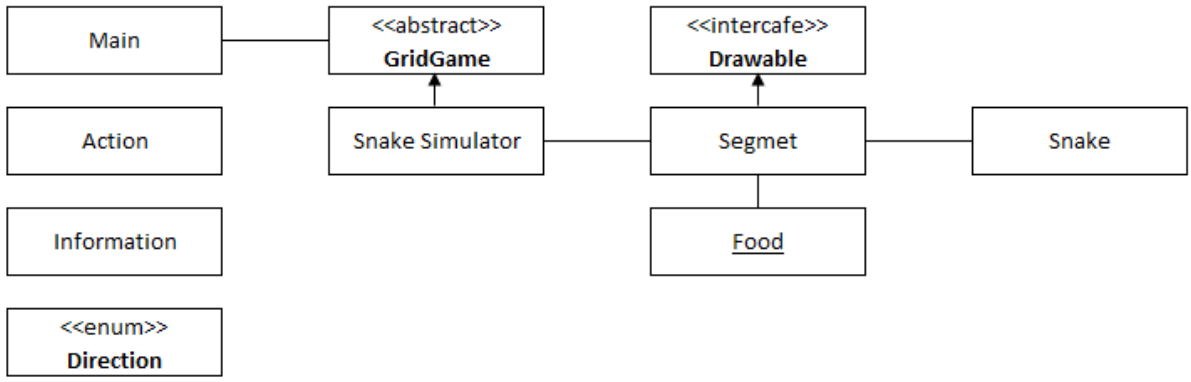
segments. SnakeSimulator is a class which has the game loop. My game runs thanks to the game loop(timerTick) in this class.

The main package has runnable code of my game.

The *elements* package includes the implementation of my creatures in this game. Segment class represents the pieces in game map. It has location information (x-y coordinates) and color information. Food is representing the food of snake. Food extends from Segment. It has only a specific color different from a segment. Snake is a Linked List of Segments. In my implementation Segments are some basic nodes (include next and previous fields) and Snake is a basic Linked List. Snake has some final fields MAX_SIZE, HEAD_COLOR, TAIL_COLOR. Also size field to store current size of a snake.

The *ai* package includes the Information and Action classes. Action classes stores the actions and their directions. Information class is the brain of my snakes in this game. It has some information about the surroundings of snake and path finder algorithm (I will make an explanation for it later.).

The *ability* package has an enum called Direction and an interface called Drawable. This package keeps the extra ability of specific classes can do.



Class Diagram

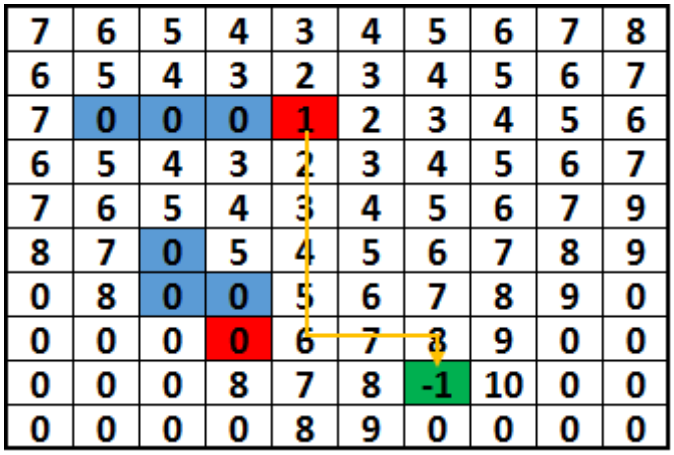
3.Information About Classes and Methods:

Action Class: It has 2 constructors. Because some actions have direction but some of them doesn't. It keeps the type of action and direction of action.

- getType() : Returns the type of action.
- getDirection() : Returns the direction of action.

Information Class: This class stores the information of surroundings of a snake, location of food, and creates the shortest path to food. Also it has information about the size of the map. Simple AI of my snakes is implemented in this class.

- getGridWidth(): Returns the width of map.
- getGridHeight(): Returns the height of map.
- getSegmentUp(): Returns the segment which is above the head of snake.
- getSegmentDown(): Returns the segment which is under the head of snake.
- getSegmentLeft(): Returns the segment which is left of the head of snake.
- getSegmentRight(): Returns the segment which is right of the head of snake.
- getFreeDirections(): Returns the list of free directions for head of a snake.
- getFreePointsAround(int x,int y, int[][] pathMatrix): It returns the list of free points around a location in map. A point is free point if it is null in segmentMap and 0 in pathMatrix(It is empty point and no action performed in path finding algorithm.) or it is -1 in pathMatrix (Location of food is -1 in pathMatrix).
- getRandomDirection(List<Direction possibleDirections): It is a static method which returns a random direction from a list of directions.
- isPoitionInsideGrid(int x, int y): It checks the position is inside the grid or not.
- getDirection(Point current, Point next): It finds and returns the direction of next point according to current point. Points have to be adjacent. If not method returns null.
- getPointsAround(int x, int y): Returns the list of all points around a location. List is not only the list of free points; it is the list of all points adjacent to a location.
- findSmallestLevelForFood(int[][]matrix): In my AI algorithm all points in the pathMatrix is a int value called path level. It represents the length of the path if snake chooses to use this point. This method finds and returns the smallest path level for the location of food.
- generatePath(): This is the key method of my simple AI algorithm. Firstly, an integer array is created and the location of snake's head (Starting point) is set to 1 and the location of food is set to -1. Then it creates a FIFO queue of points and the head of snake is added to the queue as a point object. Then a while loop starts. In this loop it polls a point from queue and creates a list of free points around that point. Then for all point in list set the value of location in the integer array as the value of polled point plus one. Then add that point to queue. This loop continues until the list of free points contains the location of food or until the queue become empty. Then if algorithm cannot find the food, it returns an empty LinkedList. If it can, it finds the smallest path level around the location of food which is greater than 0. After that until it finds the head of snake, finds the next point which has value equals to current points value minus 1. Then it adds appropriate direction to a list. When it finds the head of snake, it returns the path of snake to the food.



7	6	5	4	3	4	5	6	7	8
6	5	4	3	2	3	4	5	6	7
7	0	0	0	1	2	3	4	5	6
6	5	4	3	2	3	4	5	6	7
7	6	5	4	3	4	5	6	7	9
8	7	0	5	4	5	6	7	8	9
0	8	0	0	5	6	7	8	9	0
0	0	0	0	6	7	8	9	0	0
0	0	0	8	7	8	-1	10	0	0
0	0	0	0	8	9	0	0	0	0

An Example of Path Matrix

Segment Class: Main piece of my creatures. It has location and color information.

- setLocation(int x, int y): Sets the new location of a segment.
- setColor(Color color): Sets the color of a segment.
- getX(): Returns the x-coordinate of a segment.
- getY(): Returns the y-coordinate of a segment.
- draw(GridPanel panel): Draws a segment to the panel.

Food Class: Extends from Segment class. It has its specific color.

- generateFood(Segment[][] segmentMap): Generates a random food to an empty position. It is a static method.
- draw(GridPanel panel): Draws a food to the panel as a small square.

Snake Class: It is an implementation of a basic linked list. It uses Segments as Nodes. It keeps its head and tail. Because of the implementation of Segment, it is a doubly linked list.

- add(Segment newSegment): Adds new segment to the end of the list. Also it sets the color of segment.
- addToBegining(Segment newSegment): Adds new segment to the begining of the list. Also it sets the color of new segment as HEAD_COLOR, and sets the color of old head as TAIL_COLOR.
- removeLast(): Removes and returns the last element of the list. Firstly it keeps the tail of list. Then make the previous segment of tail the new tail of list. At the and it returns the old tail.
- generateFirstSnake(): Generates the first snake of the game. It is a static method.
- move(Direction direction): Removes the last segment of snake. Then determines the new location of it according to direction. It sets the new location of last segment and add it to the beginning of snake.

- reproduce(): Removes the last 4 segment of snake and add them to the beginning of a new snake in order. Then it returns the new snake.
- eat(Food food): It creates new segment at location of food. Then it adds the new segment at the beginning of snake.
- chooseAction(LocalInformation info): Firstly checks the size of snake. If its size equals to maximum size, it returns reproduce action. If not, it checks all segments around. If one of the is food, it returns the eat action with appropriate direction. If it cannot find food, then it checks there is a free direction or not. If there is not any free direction, it returns stay action. If there is a free direction, then it checks that new path is necessary or not. If it necessary it calls the generatePath() method from info object. Then move to the optimum direction. If all optimum directions are filled. Then tries to move a random direction.

GridGame Class: It stores timer, drawable panel and set of drawable segments. We can set the size of our world and size of squares and frame rate with the parameters of its constructor.

- redraw(): Firstly clears the panel. Then draws the grid. At the end draws all segements in the drawables set to the panel.
- addDrawble Methods: These methods add new drawable elements to the drawable set in terms of the type of element. I use polymorphism to implement this method.
- removeDrawable(Food food): Removes eaten food from drawables set.
- start(): Gives an initial delay to game when it starts. Then starts the game loop.
- stop(): Stops the game loop.

SnakeSimulator Class: It is the main class of my game. Game map and list of elements which make an action (all snakes) are stored in this class. Also the game loop (timerTick) is implemented in this class. This class extends from GridGame class. Also you can change the size of the world by changing the first two parameters of constructor of SnakeSimulator. Thrid parameter changes the size of squares. The last parameter, is called frame rate, changes the speed of the game.

- timerTick(): This method determines and executes actions for all snakes. At the beginning it copies the list of snakes, because it cannot add new snake when traversing the list with iterator (It traverses the copy one.). Then a loop is started for each snake in the list. Firstly, it copies the segmentMap then removes the current snake from map. After that snake choses its action with its instance method chooseAction. Then executes action and snake is put to

map again. How to executes selected action is the following. If action is stay it doesn't do anything. If action is move, the move method of current snake is called. If action is reproduce, new snake is created with the reproduce method of snake. Then new snake is added to map, snakes list and drawables set of parent class. If action is eat, snake will eat that food with the method eat. This method returns the new segment of snake and this segment is added to drawables set. Then food is removed from drawables. After that new food is generated and the newPathCheck of all snakes is set to true. Because location of food is changed and new path is necessary for all snakes.

- getSegmentMap(): Returns the segment map.
- createLocalInformation(Snake snake, Point foodLocation, Segment[][] segmentMapCopy): This method creates information for snakes. Firstly, it creates HashMap for segments around snake's head. Then it creates a list of free directions for snake. At the end retuns new Information object for snake.
- addElement Methods: These methods add elements to the lists and arrays in the game. If parameter is instance of food, it adds it to segmentsMap and drawables set. If parameter is instance of snake, it also adds it to snakes list.
- removeSnakeFromMap and addSnakeToMap : These methods traverses the segments of snake. Then remove or add segments to the segmentMap. (Segments are basic nodes. They have next segment field.)
- getSegmentAtPosition(int x, int y, Segment[][] segmentMapCopy): It returns the segment at the desired position. It uses a copy of segmentMap because when a snake is removed from map in the game loop, some segments are missing.
- getSegmentAtPosition(int x, int y,Direction direction, Segment[][] segmentMapCopy): It returns the segment at the desired direction. It uses getSegmentAtPoisition method to find segment.
- isPoitionInsideGrid(int x, int y): It checks the position is inside the grid or not.

ApplicationWindow Class: It creates the frame of our user interface. And create contact between frame and panel.

- initialize(): Creates the JFrame and set its bounds. It is automatically called by the constructor of this class.

GridPanel Class: This class extends from JPanel and runs as a JPanel which has a grid board on it. It provides some drawing methods suitable for pixel-like games.

- clearCanvas(): It clears all elements on the panel, then set the color of background as white.

- getGridWitdth(): Returns the width of grid.
- getGridHeight(): Returns the height of grid.
- drawGrid(): It draws the gridlines.
- drawSquare(int x, int y, Color color): Firstly it checks the coordinates x and y is in the grid or not. Then if coordinates are in the grid, paints the full square with the parameter color.
- drawSmallSquare(int x, int y, Color color): Firstly it checks the coordinates x and y is in the grid or not. Then if coordinates are in the grid, paints a small square inside of square with the parameter color.

4. Conclusion:

As you can see from the output, I solve the problem correctly. Snakes may sometimes get stuck by curling up onto themselves, near a wall or sometimes even with each other. I cannot solve this completely but I try to reduce the possibility of that by generates all path (not only the current direction). As you can see, in my implementation you can change the size of world and game speed like you expected. I use packages to organize my classes an understandable manner.