1. Introduction

Snakes is a game that we could not play, it will be playing itself. The snakes will move and hunt for food themselves without any input from the user .The game will start with only one snake(initially its size is 4), and one food which will be created randomly. A snake consists of consecutively placed blue squares and a special red square denoting the head.

Snakes are able to move in four directions (left, right, up, down) .They can not go past the world boundaries. There is a single randomly placed food (yellow color) in the game world at any time. Snakes try to find and consume the food in the game world. When a snake consumes the food, it will grow by 1.

When a snake reaches size 8 , it will divide, producing two separate snakes of size 4. The game will continue as the snakes will multiply and hunt for more food.

This game is grid-based ,so I used/adapt the codes from my Project 1 to build the game.

2. Class Hierarchy

Within the content of this project, there are 5 packages.

- **ui :** includes the classes that are necessary for creating the main interface of the game, which will enable to visualize the world.
- **game :** includes the classes that are responsible for constructing the bridge between the backend of the game and visual part.
- **main :** includes the Main class that triggers the start of the game with the necessary parameters.
- **simulator :** includes the game logic, by using Action ,NatureSimulator classes, and provides information by using LocalInformation class.
- **project :** includes the game objects (Snake,Food), which they will be shown on the game grid, and they interact each other .

ui packages has two classes.

- **ApplicationWindow :** The main application window. Initialize the JFrame with bounds x=50, y=50, width=500, height=520 parameters. (these are the parameters of frame: x is the x coordinate on the computer screen, y is the y-coordinate on the computer screen, width and height are the frame's width and frame's height) Takes a gridpanel as a parameter, and adds the gridPanel to this frame.
- **GridPanel :** A drawable panel structured as a grid. Provides some drawing methods suitable for pixel-like game entities.

game packages has three classes.

- **Direction :** is a enum, which can be UP,DOWN,LEFT,RİGHT (Enum is a special Java type used to define collections of constants)
- **Drawable** : Interface that allows objects to be drawn on a grid panel. When any class implement this interface, draw() method also should be written.
- **GridGame :** a class representing a generic grid-based game. This class is a super class of NatureSimulator class.(NatureSimulator class extends the Gridgame class)
  This class draw the drawable objects and gridlines to GridPanel. Each time tick to redraw the drawable objects added to this game.

main package has one class.

- **Main :** contains the main method(). And if we run the main class, the game will start. We can change the world width and height ,also size of each square and game speed, by changing parameters of game object.

simulator package has three classes.

- **Action :** a class representing the possible actions for Snake game. It could be four actions: Move, Reproduce ,Attack, Stay. These actions is implemented as enum Type.
- **LocalInformation :** Class representing the information a snake has about its surroundings. Automatically created and passed by the game to each creature at each timer tick. Every snake will learn the freeDirection list (which could move) and food position on the grid by using this class.
- **NatureSimulator :** Class that implements the game logic. Each timerTick, new action is executed for each Snake object at the timerTick() method. At the main method , game will created by instantiate a new natureSimulator object by giving (gridWidth, gridHeight, gridSquareSize, frameRate) parameters to this object.

project package has four classes.

- **Cell :** it is the main component of food and snake objects. Food is a object which contains only one cell, but snake has list of cells. Each cell has x and y field. These fields are the coordinates on the grid.
- **Creature :** this class is an abstract class, and it is super class of Food and Snake classes. Food and Snake classes are child class of Creature.
- **Food :** is an child class of Creature, and it contains only one cell. It also implements drawable interface and draw() method. In that cell, it will draw the square yellow color.
- **Snake :** is an child class of Creature, and it has list of Cells inside. The constructor of snake takes a list (linkedlist) of cells as a parameter and snake will be formed by these cells. It also

implements drawable interface and draw() method. The first element(cell) in the list(head) will draw the square red color ,others draw the square blue color.

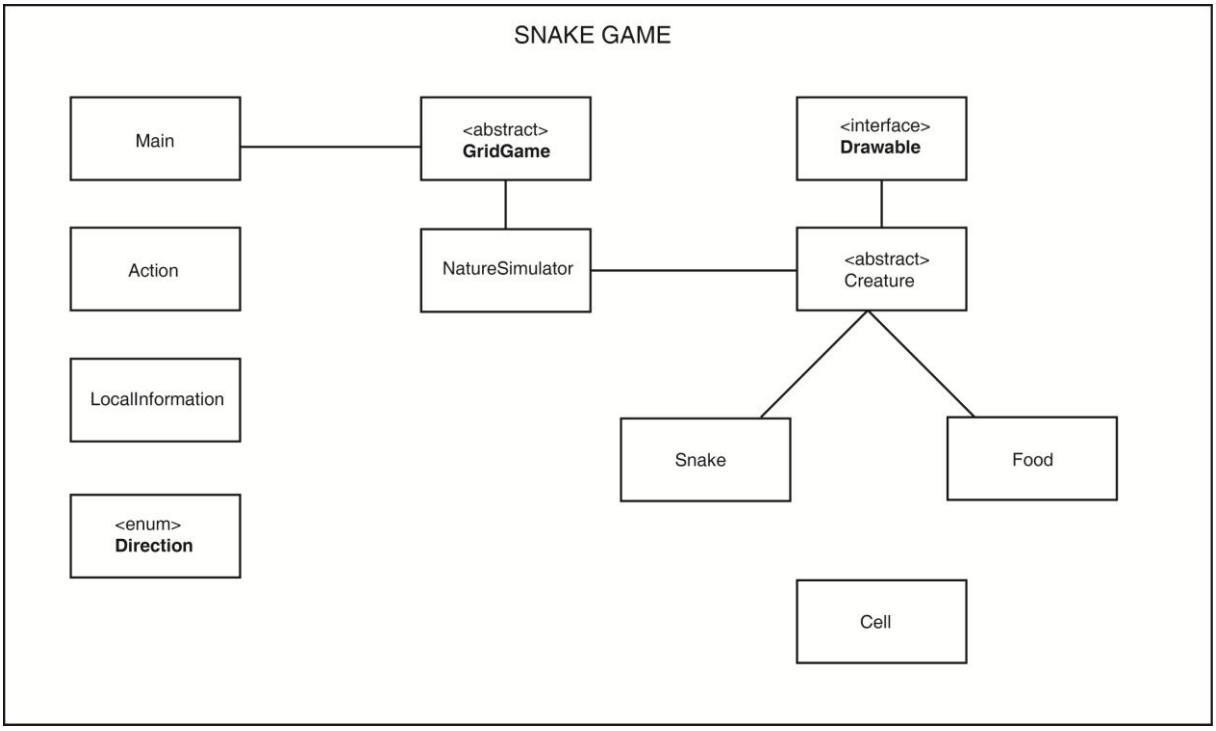

Figure 1. Class Diagram

3. Gameplay

The game is started through a call within the main class. So we should run the main class to start the game. By creating a new NatureSimulator object and sending the start message to that object, the game automatically creates the UI and executes the game. Then , it proceeds with the logic implemented in NatureSimulator.

How to run with different configurations? (changeable parameters)

In the Main class, we can change the world width and height ,also size of each square and game speed, by changing parameters of game object.

```
// You can change the world width and height, size of each grid square in pixels or the game speed

NatureSimulator  game = new NatureSimulator (width, height , size of each square,  game speed);
```

*__width :__ width of the world (changeable)

*__height :__ height of the world (changeable)

* __size of each square :__ squares(cells) pixel size (changeable)

*__game speed :__ speed of the game/frame rate (changeable)

To run the code ,we should run the main class. (this procedure mentioned above)

What is the game logic?

The main loop of the game, where each iteration is implemented by the timerTick method in NatureSimulator class, is executed via a periodic timer within the GridGame super class. The frameRate parameter provided to the NatureSimulator constructor determines the speed of the game, i.e. how many times per second timerTick will be called. The game continues to run by itself until the application window is closed.

Within each timerTick iteration, the game executes one of the possible actions for every single creature available in the game.

4. Implementation Details

Each turn forces the snakes in the game to take an action to progress the game. There are some rules that determine the actions to be taken by the snakes. This section provides the details about the implementation and rules of the game.

There are 4 possible actions that a snake can perform (enum Type in class Action):

- Move: Snake can move to an adjacent empty cell.
- Reproduce: Snake can divide i.e. create a new snake when reaches size 8. The newly born snake's head will be the old snake's tail
- Stay: Snakes can stay motionless when they unable to choose a direction to move.
- Attack: Snakes can eat food in the adjacent cell and it will grow by 1.

**Food object could not perform these 4 actions, but these methods exist in the Food class, they do not make any changes. Actually I do not need to make Creature class as a super class of Food class, but it would make easier to implement NatureSimulator class for me. (I do not need to change so many things in that class)

At each game loop iteration, the game will call the chooseAction(LocalInformation information) method of each creature, which is expected to return the action chosen by the creature(actually only snakes will chose, foods returns null) . We can think of this method as the brain of a creature: Via this method, the creature will decide on which action to perform depending on the information about its environment. This information is contained in an instance of LocalInformation class, provided automatically as an input to this method.

The return type of chooseAction(...) is Action which contains the type of the action and the direction of it, if applicable. Actions move and attack require a direction, while action reproduce(division) and stay should not have a direction. After chooseAction(...) method returns the chosen action, the game

logic implemented in NatureSimulator checks whether the chosen action is valid, and if so, calls the respective action method of the creature. The possible action methods are:

- Cell move(Direction direction)
- Creature reproduce()
- void attack(Creature attackedCreature)
- void stay()

The coordinate system of the world: Position (0,0) is the top-left square, x coordinate increases from left to right, y coordinate increases from top to bottom. (The example world created in main is 50 x 50)

How to represent snakes?

Snakes are represented by list of cells. I create a LinkedList of Cell object. And when creating a new snake, construction of new snake takes list of cells (4 cell) as a parameter. Then this cells are added to the snakePartList. The firstly added cell is head of the snake.

```
private LinkedList <Cell> snakePartList =new LinkedList <Cell> ();

public Snake(LinkedList <Cell> cells) {
		this.snakePartList=cells;
	}
```

All creatures should implement the Drawable interface and consequently the draw method, so that they can be visualized on the UI. Snake's draw method implementation is as follows:

if the cell is the head (first element in the list) of the LinkedList , it will shown by Red color, but others will be shown by Blue color.

```
	public void draw(GridPanel panel) {
		for(Cell cell : snakePartList) {
			if( snakePartList.getFirst()==cell) {
				panel.drawSquare(cell.getX(), cell.getY(), Color.RED);
			}
			else  {
				panel.drawSquare(cell.getX(), cell.getY(), Color.BLUE);
			}

		}
	}
```

How handled movement of snakes ?

In the snake class, I write move() method, which takes a parameter of directionToMove.
In this method, we find the coordinates of first cell (head) of snake. Then we create a new Cell by checking the condition of four direction (by checking parameter directionToMove).

Then we add the next Cell (which head will move) to the snake cell List, by changing the head. (moved cell is added to head of snakePartList)
Then the cell at the tail of the snake is removed from the snake's cell list (snakePartList).

So snake will move the next position(direction), which is chosen at choseAction method.

```
public Cell move(Direction direction) {
		Cell node=null;
		int headX=this.snakePartList.getFirst().getX();
		int headY=this.snakePartList.getFirst().getY();

		switch(direction) {
		case UP :
			node = new Cell(headX, headY - 1);
			break;
		case RIGHT :
			node = new Cell(headX + 1, headY);
			break;
		case DOWN :
			node = new Cell(headX, headY + 1);
			break;
		case LEFT :
			node = new Cell(headX - 1, headY);
			break;
		}
		this.snakePartList.addFirst(node);
		return snakePartList.removeLast();

	}
```

How handled dividing of snakes ?

When a snake reaches size 8 , it will divide , producing two separate snakes of size 4. The newly born snake's head will be the old snake's tail.

At the Snake class, chooseAction () method will choose, whether snake will reproduce, eat(attack), move or stay motionless .Reproduce has a priority, so when a snake reaches size 8 , it will divide immediately.

At the Snake class, I write reproduce() method. This method will work, when choseAction() method will return reproduce action, and NatureSimulator .timerTick() method will work reproduce() method of this snake.

This method(reproduce) will work as follows:

We add the old snake's last four cell to newly created list. Then we remove the old snake's last four cell. Then we create a new Snake, by giving this newly created list as a parameter of Snake. (the tail of old snake will be head of the new snake, because we add the tail firstly to the list)

```
	public Creature reproduce() {
		LinkedList<Cell> list=new LinkedList();
		LinkedList<Cell> newList=new LinkedList();

		list=this.getBody();
```

```
                //old snake's last 4 cell will be added to newList.
                for(int i=0 ; i<4 ; i++) {
                        //old snake's tail will be head of the newly created snake
                        newList.add(list.getLast());
                        list.removeLast();
                }
                //removes the last four cell of the old snake
                this.snakePartList.removeAll(newList);

                //then the new snake will be build by newList.
                return new Snake(newList);

        }
```

What is the AI logic of snakes, How could they find the food ?

First of all we choose to move ,when snake could not divide, and there will be freeDirections on the grid. (that information comes from LocalInformation class)

We should learn the snake's head coordinates by getX and getY methods. And we learn the food coordinates by the information comes from LocalInformation class)

Then we will check, whether snake is left or right of the food coordinate, and whether snake is up or down of the food coordinate. Then we add the two option(direction) to the newly created ArrayList. (named: DirectionToFood)

Then we will compare freeDirection list (which is coming from as a parameter from the LocalInformation class) with the DirectionToFood list. if they has same directions in the list, we add these directions to another Arraylist named directionToGo.

Then finally, we check whether is directionToGo has an element, or it is empty. If it has more than one element inside, we should use getRandomDirection() method of LocalInformation class to choose one possible direction . We return this direction as directionToMove, and return Action Move to the NatureSimulator class.

If the directionToGo list has no element inside, we should send the freeDirection list as a parameter to the getRandomDirection() method and it will return one direction to us.

```
                //Learn the food coordinates
                int headX=this.snakePartList.getFirst().getX();
                int headY=this.snakePartList.getFirst().getY();

                //it will adds the directions,which snake should move,to find the food
                if(headX < foodcorX ) {
                        directionToFood.add(Direction.RIGHT);
                }

                if(headX > foodcorX ) {
                        directionToFood.add(Direction.LEFT);
                }


                if(headY < foodcorY) {
                        directionToFood.add(Direction.DOWN);
                }
```

```
                        if(headY > foodcorY) {
                                directionToFood.add(Direction.UP);
                        }


                        for(int i=0; i<directionToFood.size() ;i++) {
                                if(freeDirections.contains(directionToFood.get(i)) ){
                                        directionToGo.add(directionToFood.get(i));
                                }
                        }

                        //if directionToGo list has one or more element in the list,we will
select the direction randomly by getRandomDirection method.
                        if(!directionToGo.isEmpty()) {

        directionToMove=LocalInformation.getRandomDirection(directionToGo);
                        }
                        // if directionToGo list is empty, snake will move only to
freeDirection.
                        else{

        directionToMove=LocalInformation.getRandomDirection(freeDirections);
                        }

                        return new Action(Action.Type.MOVE,directionToMove);
```

How the snake do not crash with its own body? (can not move its own body)

To check the crash condition, I write a checkCrash method in the Snake Class. checkCrash() method checks whether nextCell is in the snake's own body, by using for loop. It will compare the nextCell (parameter) x coordinate and y coordinate with the snake's cells coordinates. if they are same it will return true ,else it will return false.

Then in the NatureSimulator class , List of freeDirections is checked by using this method. Then it will send this information as a a parameter to snake object.(choseAction method will choose freeDirection from this list)

```
public boolean checkCrash(Cell nextCell) {
                for (Cell cell : snakePartList) {
                        if (cell.getX() == nextCell.getX() && cell.getY() ==
        nextCell.getY()) {
                                return true;
                        }
                }

                return false;
        }
```

2013400258 - YİĞİT SARIOĞLU
SNAKE GAME - PROJECT REPORT

How the snake do not crash another snakes? (can not move over another snakes)

In the NatureSimulator class, when we find freeDirection list(arraylist) for sending this information to every snake object. And every snake's choseAction method use this information by to move empty cell.

In this method (`createLocalInformationForCreature`), first of all we should learn the snake's coordinates by getX and getY methods.

Then we create a new empty list that should store all the cells which is occupied by snakes body. We are looking all snakes in the creature list(which was added when new creature was created) , then we add these snake's cells to the newly created LinkedList named cellList.

After we compare the snake's adjacent cells with the cellList. if cellList contains any of up,down,right or left cell of the current snake, the boolean equation should be false. And this condition will be checked when adding new Direction to freeDirection list.

```
int x = creature.getX();
int y = creature.getY();

//This was an AI algorithm which prevents snakes move over another snakes.
LinkedList <Cell> cellList=new LinkedList ();
for( Creature snakes :this.creatures ) {
	//add the all snakes(in the grid) list of cells(body) to the cellList
	if(snakes instanceof Snake) {
		cellList.addAll(snakes.getBody());

	}
}

boolean upfield=true;
boolean downfield=true;
boolean rightfield=true;
boolean leftfield=true;
//And checks whether this cells are occupied by another snake
for(int i=0 ; i<cellList.size() ; i++ ) {
	if(cellList.get(i).getX()==x && cellList.get(i).getY()==y-1) {
		upfield=false;
	}
	if(cellList.get(i).getX()==x && cellList.get(i).getY()==y+1) {
		downfield=false;
	}

	if(cellList.get(i).getX()==x-1 && cellList.get(i).getY()==y) {
		leftfield=false;
	}
	if(cellList.get(i).getX()==x+1 && cellList.get(i).getY()==y) {
		rightfield=false;
	}

}
```

2013400258 - YİĞİT SARIOĞLU
SNAKE GAME - PROJECT REPORT

How the food is created randomly in empty cell?

I write a createFood() method in the NatureSimulator class, to do this work. I have used do/while loop to find empty random cell. isPositionOk(x,y) method looks if is there any other snake object occupied the cell. If there is a snake at that cell ,method returns false so it goes in to loop again and Random method returns another x,y coordinates. After the empty position is found, newly food object (at that cell) is created and added to the game by addCreature() method.

```
public void createFood() {

        int x,y;

        //Makes random positions/coordinates
        do {
                x = new Random( ).nextInt(this.getGridWidth());
                y = new Random( ).nextInt(this.getGridHeight());
        } while ( isPositionOk(x,y)==false);
 //Check whether that coordinates are free.
        foods = new Food(new Cell(x,y));

        addCreature(foods);  // add this food

}
```

** addCreature(Creature) method could add both snakes and foods in to the game

** removeCreature(Creature) method could remove foods from the game.