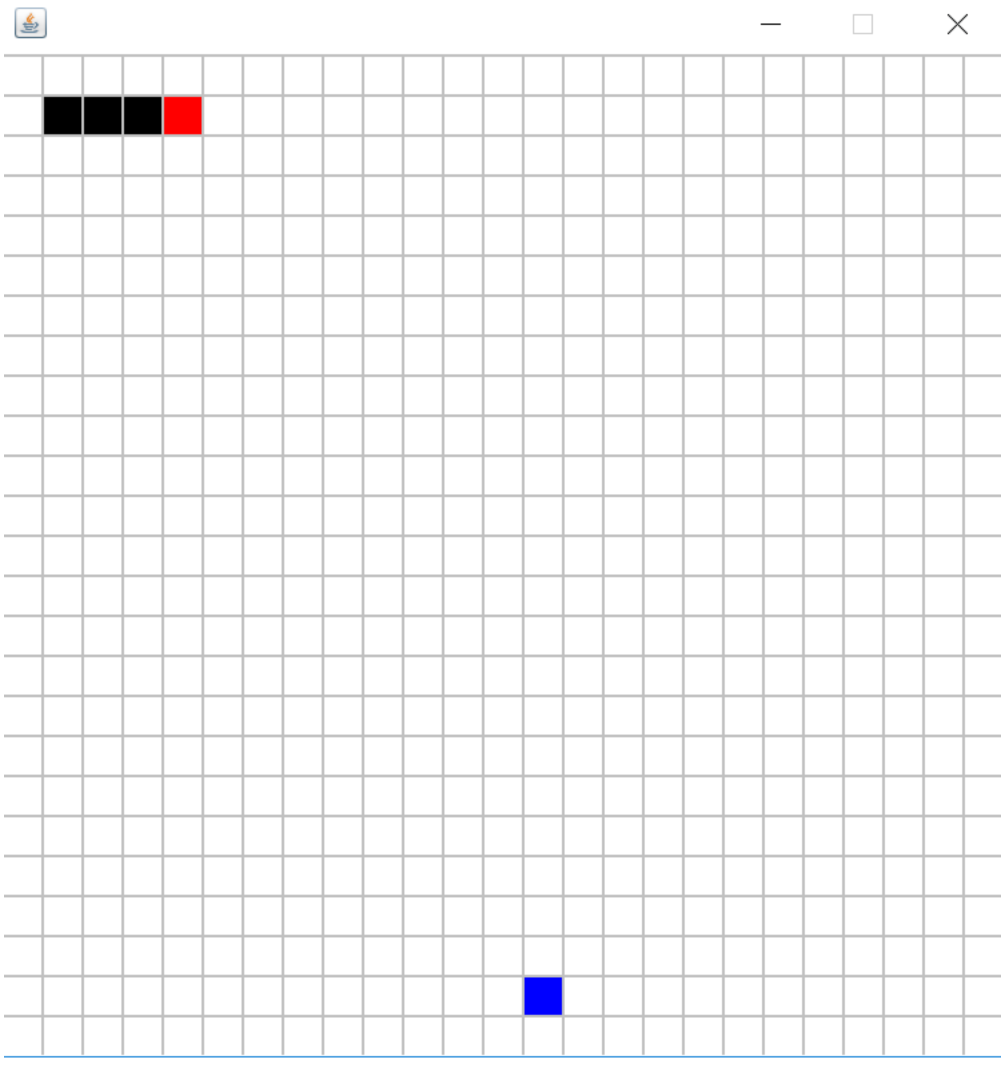
# CmpE 160 - Introduction to Object Oriented Programming

# Project #2 - Snakes!

## Introduction

For this project, we implemented a game called Snakes! It is a game like the classic game Snake with some important differences and enhancements.

In this project, a snake consists of consecutively placed black squares and a special red square denoting the head. Food's color is blue.



## Class Hierarchy

Thereare 5 packages in the project:

The **ui** package includes the classes that are necessary for creating the main interface of the game.

The **game** package includes the classes that are responsible for constructing the bridge between the backend of the game and the visual part. It provides a generic game system that can be used for any grid-based game.

The **main** package includes the Main class that triggers the start of the game with the necessary parameters. The Main class contains example code to create a UI with the specified grid world size and game speed, and adds several randomly positioned creatures to the game world.

The **project** package consists of the information of creatures .Snake and Food classes are in this package.

Lastly, the main logic and operations of the game are partially implemented in the classes that are provided by the **naturesimulator** package.

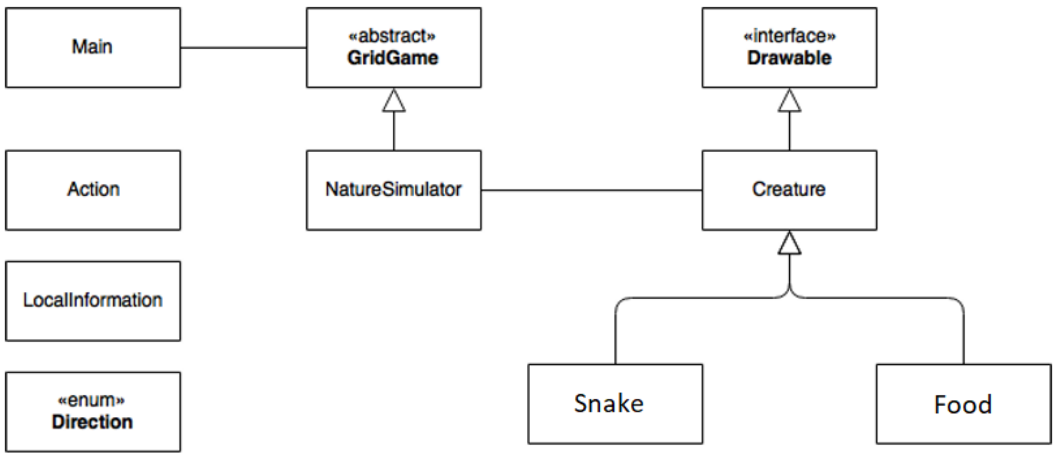


Figure 2: Class diagram.

## Gameplay

The game is started through a call within the Main class.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.

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 superclass. 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.

## Implementation Details

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

• Move: Snakes can move to an adjacent empty cell.

• Reproduce: Snakes can reproduce.

• Stay: Snakes can stay at the same space.

• Attack: Snakes can attack food, in which case it will kill it and move to its position.

At each game loop iteration, the game will call the chooseAction (LocalInformation information) method of each creature. We can think of this method as the brain of a creature: Via this method, the snake 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.

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:

• void move(Direction direction)

• Creature reproduce(int x , int y)

• void attack(Creature attackedCreature)

• void stay()

## Methods of Snake.class

```
/**
* Via this method,the Snake will decide on which action to perform depending
on the information about its environment.
* @param information representing the information a Snake has about its
surroundings
* @return action the action chosen by the Snake
*/
@Override
public Action chooseAction(LocalInformation information, Creature
Food_location) {
if (this == snake1.getFirst()) {
// All of directions which are available for move
ArrayList<Direction> directionforFood = new ArrayList<>();

if (Food_location != null) {
// if x minus y equal to 1 or -1, then it can attack
if ((getX() == Food_location.getX()
&& (getY() - Food_location.getY() == 1 || getY() - Food_location.getY() == -1))
|| (getY() == Food_location.getY()
&& (getX() - Food_location.getX() == 1 || getX() - Food_location.getX() == -1))) {
//// ATACKKKKK
if (information.getCreatureDown() instanceof Food) {
return new Action(Type.ATTACK, Direction.DOWN);
}
if (information.getCreatureUp() instanceof Food) {
return new Action(Type.ATTACK, Direction.UP);
}
if (information.getCreatureRight() instanceof Food) {
return new Action(Type.ATTACK, Direction.RIGHT);
}
if (information.getCreatureLeft() instanceof Food) {
return new Action(Type.ATTACK, Direction.LEFT);
}

}
// find coordinate between Food and head of snake
if (getX() < Food_location.getX()) {

directionforFood.add(Direction.RIGHT);
}
if (getX() > Food_location.getX()) {

directionforFood.add(Direction.LEFT);
}
if (getY() > Food_location.getY()) {

directionforFood.add(Direction.UP);
}
if (getY() < Food_location.getY()) {

directionforFood.add(Direction.DOWN);
}
}
// direction is equal to one of freeDirection spaces
Direction direction =
LocalInformation.getRandomDirection(information.getFreeDirections());
if (directionforFood.size() != 0) {
//

Direction FoodDirection =
LocalInformation.getRandomDirection(directionforFood);
////
```

```
if (FoodDirection == Direction.RIGHT && information.getCreatureRight() ==
null) {
return new Action(Type.MOVE, FoodDirection);
} else if (FoodDirection == Direction.LEFT && information.getCreatureLeft() ==
null) {
return new Action(Type.MOVE, FoodDirection);
} else if (FoodDirection == Direction.UP && information.getCreatureUp() ==
null) {
return new Action(Type.MOVE, FoodDirection);
} else if (FoodDirection == Direction.DOWN && information.getCreatureDown()
== null) {
return new Action(Type.MOVE, FoodDirection);
} else {
// if head cannot move, then head go to free direction

return new Action(Type.MOVE, direction);
}
}

return new Action(Type.STAY);
}

return null;
}





/**
          * it provides that all Snakes can be visualized on the UI
     @param panel the properties of Grid
          */
         @Override
         public void draw(GridPanel panel) {
                  panel.drawSquare(getX(), getY(), Color.RED);
                  for (int i = 1; i < snake1.size(); i++) {
                           panel.drawSquare(snake1.get(i).getX(), snake1.get(i).getY(),
Color.BLACK);
       }}
```

```
/**
 * Creating a new Snake of the same type on an adjacent empty space
 * @param direction  representing the empty space where
   surroundings of a Snake
 * @return a new Snake
 */
@Override
public Creature reproduce(int x, int y) {

         int headx = snake1.getFirst().getX();
         int heady = snake1.getFirst().getY();


         snake1.add(1, new Snake(headx, heady));
         snake1.getFirst().setX(x);
         snake1.getFirst().setY(y);
          head=snake1.getFirst();
//if parts of snakes equal to eight, then it will divide
         if (snake1.size() == 8) {
    //save the coordinates of  last element and delete it
                  int lastsnakex = snake1.getLast().getX();
                  int lastsnakey = snake1.getLast().getY();

                  snake1.removeLast();
                  x1 = snake1.getLast().getX();
                  y1 = snake1.getLast().getY();
                  snake1.removeLast();
                  x2 = snake1.getLast().getX();
                  y2 = snake1.getLast().getY();
                  snake1.removeLast();
                  x3 = snake1.getLast().getX();
                  y3 = snake1.getLast().getY();
                  snake1.removeLast();

         return new Snake(lastsnakex, lastsnakey);
         }
         return null;
}
```

```
/**
	 * Changing the position of the Snakes
	 * @param direction modifying its position to the direction which is an
adjacent empty cell
	 */
	@Override
	public void move(Direction direction) {
		// get first element
		int headX = snake1.getFirst().getX();
		int headY = snake1.getFirst().getY();
		// change last element with first element
		snake1.getLast().setX(snake1.getFirst().getX());
		snake1.getLast().setY(snake1.getFirst().getY());
		// the last element equals to the first element
		snake1.set(0, snake1.getLast());

		//move the head
		if (Direction.RIGHT == direction) {

			head.setX(headX + 1);

		} else if (Direction.LEFT == direction) {

			head.setX(headX - 1);

		} else if (Direction.DOWN == direction) {

			head.setY(headY + 1);

		} else if (Direction.UP == direction) {

			head.setY(headY - 1);

		}

		snake1.addFirst(head);
		snake1.removeLast();	}
```

```
/**
          * Snakes can attack another adjacent foods,in which case it will
            kill(remove)
          * it and move to its position
          * @param attackedCreature
          */
         @Override
         public void attack(Creature attackedCreature) {

                  attackedCreature.setHealth(0.0);
         }
```

## Methods of NatureSimulator class

```
/**
* Class that implements the game logic for Nature Simulator.
*
*/
public class NatureSimulator extends GridGame {

private List<Creature> creatures;
private Creature[][] creaturesMap;
private Creature attackedCreature = null;

/**
* Creates a new Nature Simulator game instance
*
* @param gridWidth number of grid squares along the width
* @param gridHeight number of grid squares along the height
* @param gridSquareSize size of a grid square in pixels
* @param frameRate frame rate (number of timer ticks per second)
*/
public NatureSimulator(int gridWidth, int gridHeight, int gridSquareSize, int
frameRate) {
super(gridWidth, gridHeight, gridSquareSize, frameRate);
creatures = new ArrayList<>();
creaturesMap = new Creature[gridWidth][gridHeight];
}
```

```
@Override
protected void timerTick() {
// Determine and execute actions for all creatures
ArrayList<Creature> creaturesCopy = new ArrayList<>(creatures);
for (Creature creature : creaturesCopy) {
if (creature.getHealth() <= 0.0) {
// Creature is dead, hence continue with the next creature
// (It must be already removed, it won't be in the list for the next tick)
continue;
}
if (creature instanceof Food) {
attackedCreature = creature;
}
// Reset creature's map position (its position will be marked again, if it still
lives)
updateCreaturesMap(creature.getX(), creature.getY(), null);
if (creature instanceof Snake)
for (int i = 1; i < ((Snake) creature).snake1.size(); i++) {
updateCreaturesMap(((Snake) creature).snake1.get(i).getX(), ((Snake)
creature).snake1.get(i).getY(),
((Snake) creature).snake1.get(i));
}
// Choose action
Action action =
creature.chooseAction(createLocalInformationForCreature(creature),
attackedCreature);
// clear CreaturesMap
if (creature instanceof Snake) {
for (int i = 1; i < ((Snake) creature).snake1.size(); i++) {
updateCreaturesMap(((Snake) creature).snake1.get(i).getX(), ((Snake)
creature).snake1.get(i).getY(),
null);
}
//only head can do action
if (creature == ((Snake) creature).snake1.getFirst()) {
// Execute action
if (action != null) {
if (action.getType() == Action.Type.STAY) {
// STAY
} else if (action.getType() == Action.Type.MOVE) {
```

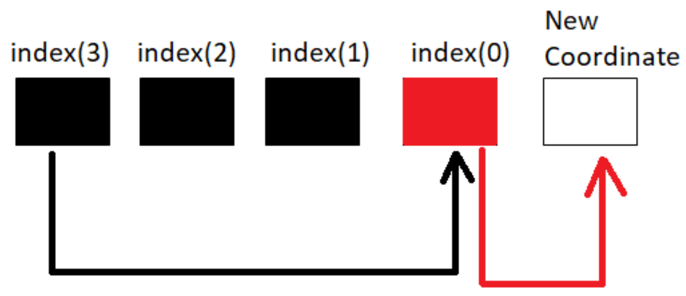
```
// MOVE
if (isDirectionFree(creature.getX(), creature.getY(), action.getDirection())) {
creature.move(action.getDirection());
}
} else if (action.getType() == Action.Type.ATTACK) {
// ATTACK
//first attack then reproduce
if (attackedCreature != null) {
creature.attack(attackedCreature);
/// attackk
removeCreature(attackedCreature);
updateCreaturesMap(attackedCreature.getX(), attackedCreature.getY(), null);
addCreature(creature.reproduce(attackedCreature.getX(),
attackedCreature.getY()));

if (creature instanceof Snake) {
for (int i = 0; i < ((Snake) creature).snake1.size(); i++)
creaturesMap[((Snake) creature).snake1.get(i).getX()][((Snake)
creature).snake1
.get(i).getY()] = ((Snake) creature).snake1.get(i);
}
addFood();
}}
if (creature instanceof Snake) {
for (int i = 1; i < ((Snake) creature).snake1.size(); i++)
creaturesMap[((Snake) creature).snake1.get(i).getX()][((Snake)
creature).snake1.get(i)
.getY()] = ((Snake) creature).snake1.get(i);
}
updateCreaturesMap(creature.getX(), creature.getY(), creature);
}
}}
if (creature.getHealth() <= 0.0) {
// Remove creature if its health is zero
removeCreature(creature);
} else {
// Mark current creature's map position, if it still lives

// fill creaturesmap
if (creature instanceof Snake) {
```

```
for (int i = 1; i < ((Snake) creature).snake1.size(); i++)
creaturesMap[((Snake) creature).snake1.get(i).getX()][((Snake)
creature).snake1.get(i)
.getY()] = ((Snake) creature).snake1.get(i);
}
updateCreaturesMap(creature.getX(), creature.getY(), creature);
}}}
```

## SNAKE MOVEMENT(Linked List)



**ALİ BATIR - 2015400261**