Nibbles

NibblesGameFrame

```
- rows: int
- cols: int
- size: int
- winCondition: int
- board: BoardPanel
- gameTable: NibblesGameStructure
- extraWalls: ArrayList<Coordinate>
```

```
+ NibblesGameFrame(title: String, rows: int, cols: int, size:int)
+ NibblesGameFrame(title: String, rows: int, cols: int)
+ loadGraphicsFile(file: File)
# processKey(e: KeyEvent)
# animateInit()
# animateNext()
# animateFinal()
- gameSettings()
- printNextPhase()
- win()
- checkWallFile(wallPosition: ArrayList<Coordinate>): boolean
- printSnake()
- printFood()
- printWalls()
```

NibblesGameStructure

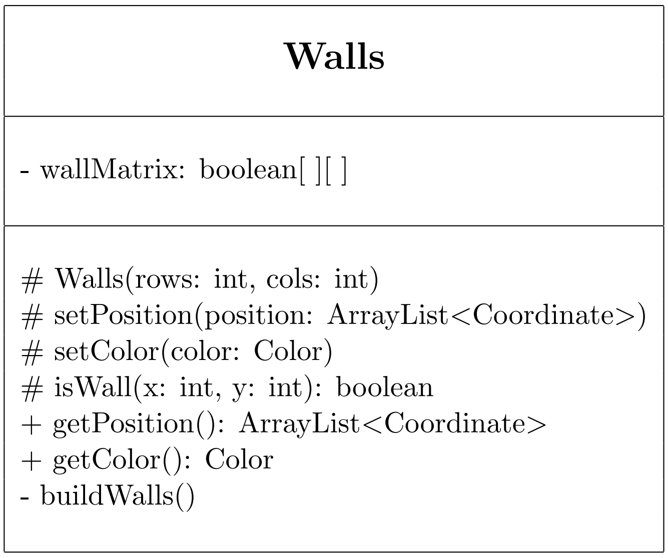
```
# snake: Snake
# walls: Walls
# food: Food
# gameOver: boolean
# initialSafeSpace: int
- rows: int
- cols: int
- growthFactor: int
- depth: int
- points: int
- changing: boolean
```

```
+ NibblesGameStructure(rows: int, cols: int, extraWalls: ArrayList<Coordinate>)
+ getSnakeDirection(): int
+ getSnake(): ArrayList<Coordinate>
+ getFood(): Coordinate
+ getWalls(): ArrayList<Coordinate>
+ getPoints()
# changeSnakeDirection(direction: int)
# nextStep()
- setBoardObjects(extraWalls: ArrayList<Coordinate>)
- setSnake()
- randomSnakeTail(): Coordinate
- setFood()
- setWalls()
- addNewWalls(position: ArrayList<Coordinate>)
- checkNextSlot(x: int, y: int)
- buildNextPhase(headX: int, headY: int, TailX: int, TailY: int)
```

BoardObjects
color: Color # position: ArrayList<Coordinate>
setPosition(position: ArrayList<Coordinate>) # *setColor(color: Color)* + *getPosition*: ArrayList<Coordinate> + *getColor*: Color

Snakes
headColor: Color # direction: int # lenght: int

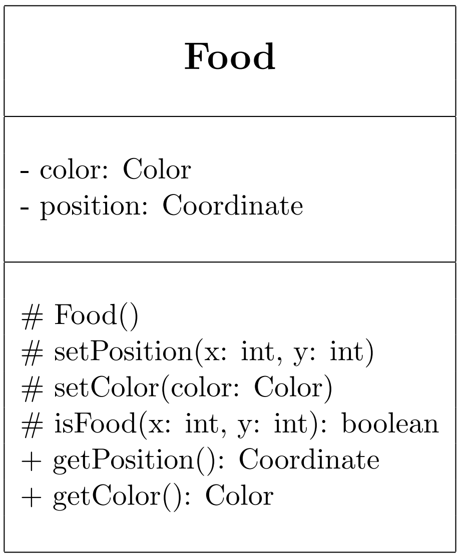
# Snakes(color: Color) 	
# setPosition(position: ArrayList<Coordinate>) 	
# setColor(color: Color) 	
# setLenght(lenght: int) 	
# isSnake(x: int, y: int): boolean 	
# getHead(): Coordinate 	
# getTail(): Coordinate 	
# moveSnake(x: int, y: int, isGrowing: boolean) 	
# moveSnake(x: int, y: int) 	
# changeDirection(direction: int) 	
+ getPosition(): ArrayList<Coordinate>
+ getColor(): Color
+getLenght(): int



Walls

```
- wallMatrix: boolean[ ][ ]
```

```
# Walls(rows: int, cols: int)
# setPosition(position: ArrayList<Coordinate>)
# setColor(color: Color)
# isWall(x: int, y: int): boolean
+ getPosition(): ArrayList<Coordinate>
+ getColor(): Color
- buildWalls()
```



Food

```
- color: Color
- position: Coordinate
```

```
# Food()
# setPosition(x: int, y: int)
# setColor(color: Color)
# isFood(x: int, y: int): boolean
+ getPosition(): Coordinate
+ getColor(): Color
```

I've worked with two main classes: one taking care of the graphical aspects of the game (`NibblesGameFrame`*), relying on the other one (*`NibblesGameStructure`*), which is the game engine. The first one is a subclass of the given* `KeyAnimationBoardFrame` *class. The latter is an independent one, relying in turn to three other classes:* `Walls`*,* `Snakes` *and* `Food`*. I decided to make the first two of these, subclasses of the abstract class* `BoardObject`*, while leaving* `Food` *independent. The reason is that the common methods require different arguments for* `Food` *and the same ones for the other two.*

`GameSounds` *is just an extra class which adds some music and sound effects.*

`NibblesGameFrame`

Methods *animateInit* and *gameSettings* starts the game after giving the user some infomations and choices. The variable **gameTable** of type `NibblesGameStructure` is here initialized. This variable is meant to be used during the whole process, since is the one collecting all the data on the current game. This variable can be inizialised with a specific frame dimension and walls disposition, given externally through the method *loadGraphicsFile.*

animateNext method uses the variable **gameTable** to produce the next phase of the game and then prints it with method *printNextPhase* (a collection of all printing methods). It also checks the win and lose conditions at each step.

`NibblesGameStructure`

The class Constructor sets the "game dimensions" and initializes the three board objects (a snake, walls and a food package). First, the walls around the frame are built (eventually with extra internal walls), then the snake is placed randomly with the use of *randomSnakeTail* method and **initialSafeSpace** variable. As the names say, the position of the tail is choosen randomly (with regards of the walls position), then the method checks if there is enogh space to place a snake (vertically) with an appropriate initial safe space (the lenght of the snake is stored in a `Snakes` class variable).

Class `Snakes` has a variable **direction** (set "down" at the beginning). At each step, the snake moves in this direction, which can be changed by the method *changeSnakeDirection.* When directional arrows are pressed on the keyboard during the game, the method *processKey* of the GUI class

`NibblesGameFrame`, calls the method *changeSnakeDirection* through the variable **gameTable**. Methods *nextStep* and *buildNextPhase* take care of producing the "new snake", considering the direction.

Variable **changing** is used to prevent directional changes overlapping with one another: if a change is made, the variable is set to true in *changeSnakeDirection* method, and this method cannot be called again, until the variable is set to false, in *nextStep*, after the following phase is built.

Finally, when *buildNextPhase* is called, *checkNextSlot* is called aswell. This method checks if there is something in the next slot. If this is the case, a wall or the snake lead to a game over, while food increases **points** by one and **depth** by five, and a new food package is generated. When depth is bigger then 0, in *buildNextPhase* method, the tail is mantained in the same position and depth is decreased by one instead.

`Other Classes`

As said before, `Snakes`, `Walls` and `Food` are the game objects. Thet simply hold some informations on position and color, with setters and getters to work with these data. `Snakes` has some extra methods and variables, like the ones to change or get the direction of the snake.

P.S.

`GameSounds`

I simply did some copy-paste here and there from the internet, trying to make the codes that I've found work with my project. I don't take much credit on that part.

Riccardo Malandruccolo