**МИНЕСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное  
образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

*КАФЕДРА ТЕОРЕТИЧЕСКОЙ И ПРИКЛАДНОЙ ИНФОРМАТИКИ*

Расчетно-графическая работа

По дисциплине “Системное и прикладное программное обеспечение”



Факультет: ПМИ

Группа: ПМИ-82

Студент: Губенко.К.О.

Преподаватель: Неделько В. М.

Новосибирск

2019

**Тема: Приложение с графическим интерфейсом** *Цель: Научиться использовать средства графического интерфейса и грамотно разбивать на модули программу.*

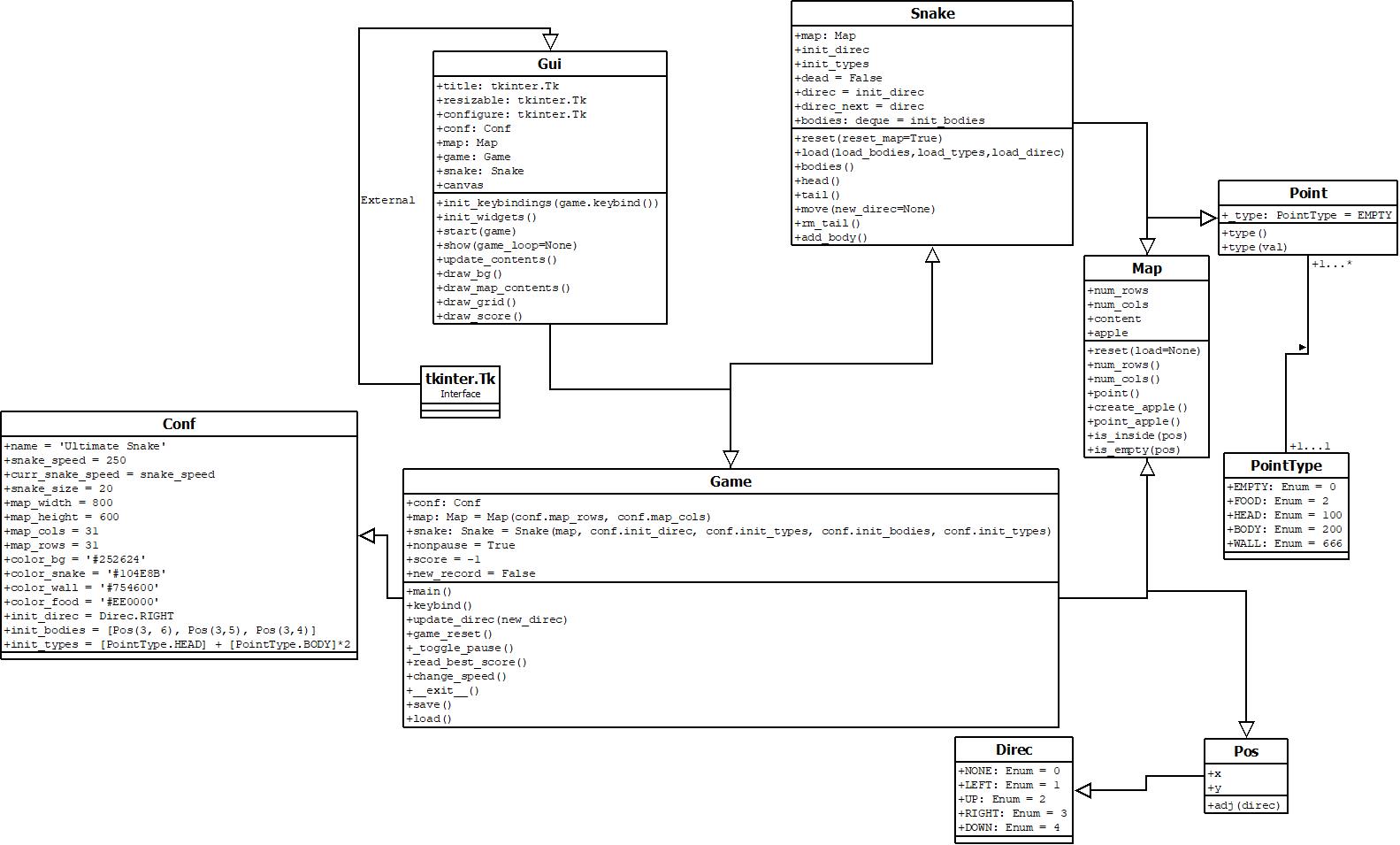
**1. Спецификация**

Техническое задание: разработать игру «Змейка», используя Python и в частности библиотеку Tkinter. При чем, с некоторыми отличительными особенностями:

* Возможность паузы
* Лучший счёт
* Возможность сохранять игру и, соответственно, загрузить игру
* Возможность перезапуска игры

**2.Архитекура, технические решения**

UML – иерархия классов:

**

Текст программы разбит на следующие модули и классы:

* Map
  + class Map
  + class PointType(Enum)
  + class Point
  + class Direc(Enum)
  + class Pos
* Snake
  + Snake
* Game
  + Conf
  + Game(Conf)
* Gui
  + class Gui(tkinter.Tk)
* SnakeGame (запускает игру)

Назначение классов и их методов:

Pos:

Нужен для хранения позиции сегментов карты. Для инициализации объекта принимает 2 параметра – координаты.

adj(self, direc) – параметр direc – направление движения змейки. Нужен для движения змейки. Возвращает следующие координаты для «головы» змеи.

Direc:

Класс типа Enum. Содержит возможные направления движения: None, Left, Up, Right, Down.

PointType:

Класс типа Enum. Содержит возможные типы точек: Empty, Wall, Body, Head, Food.

Point:

Класс точки. Изначально присваивает всем объектам тип Emty.

type(self) – getter типа

type(self, val) – setter типа

Map:

Класс карты, реализуемый в виде двумерной матрицы. Для инициализации требует 2 параметра: размерность поля – матрицы.

reset(self, load=False) – параметр load указывает происходит загрузка сохраненной игры или начало новой. Присваивает всем точкам тип Empty. Если игра загружается, то соответствующей координате присваивается тип Food. Соответствующим точкам присваивается тип Wall.

num\_rows(self) – возвращает количество строк.

num\_cols(self) – возвращает количество столбцов.

point(self, pos) – возвращает элемент матрицы, индекс которого передается параметром pos.

create\_apple(self) – создает яблоко, если его нет, путем присваивания какой-либо подходящей точке тип Food. В таком случае вернёт True. Иначе False.

point\_apple(self) – возвращает координату точки типа Food.

is\_empty(self, pos) – возвращает True, если змейка находится в пределах поля, ограниченного стенами (точками типа Wall) и не произошло «самопоедание». False в обратном случае.

Snake:

Класс самой змейки. Змейка реализована в виде нескольких сегментов типа Body и Head соответственно. Для инициализации объекта требует объект класса Map, изначальные координаты сегментов змеи, их количество и типы, также изначальное направление.

reset(self, reset\_map=True) – происходит инициализация атрибутов, используя передаваемые параметры. Если reset\_map равняется True, то запускается метод map.reset() без параметров. Независимо от параметра, присваивает соответствующим точкам типы Body, Head.

load(self, load\_bodies, load\_types, load\_direc) – изменяет змейку (координаты ее сегментов и их количество и направление движения) на сохранённые. Запускает метод map.reset(True). Присваивает соответствующим точкам типы Body, Head.

bodies(self) – возвращает список (очередь), в котором хранятся все сегменты змейки

len(self) – возвращает длину змейки (количество сегментов)

tail(self) – возвращает последний сегмент (хвост).

rm\_tail(self) – удаляет хвост змейки (присваивает тип Empty)

move(self, new\_direc=None) – движение змейки и проверка условия продолжения игры с помощью map.is\_empty(new\_head), где new\_head – координаты следующей точки по направлению движения. Реализовано движение как присваивание очередному сегменту координаты следующего.

add\_body(self) – добавление сегмента типа Body к змейке (с конца).

Conf:

Класс конфигурации, включает в себя такие параметры, как: цвета объектов, размер змейки, координаты ее появления, размеры поля и матрицы и т.д.

Game:

Для инициализации объекта требуется объект класса Conf.

main(self) – главная функция, вызывает snake.move(), map.create\_apple().

keybind(self) – возвращает кортеж кортежей названий кнопок и соответствующих функций.

update\_direc(self, new\_direc) – меняет направление движение змейки.

game\_reset(self) – присваивает изначальные значения счета и скорости змейки. Вызывает метод snake.reset()

\_toggle\_pause(self) – постановка паузы

read\_best\_score(self) – считывание из файла лучшего счета

change\_speed(self) – изменяет скорость змейки

\_\_exit\_\_(self) – выход из программы

save(self) – запись в файл необходимых для сохранения игрового прогресса данных

load(self) – считывает данные из файла и присваивает их соответствующим полям. Вызывает метод snake.load(load\_bodies, load\_types, load\_direc).

Gui:

Класс, который отвечает за графический интерфейс. Является child-классом класса tkinter.Tk.

start(game) – статичный метод, в котором происходит инициализация объекта Gui. Параметр – экземпляр класса Game. Также вызывается функция show(game.main).

show(self, game\_loop=None) – параметр – это основная функция игры, в частности game.main. Если нет паузы и игра не была проиграна, то вызывается метод обновления графики, а именно: update\_contents().

init\_keybindings(self, keybindings) – обработчик нажатий с помощью bind.

init\_widgets() – вызывается при инициализации объекта Gui. Выполняет функцию конфигурации виджетов, то есть для отображения в окне канваса, кнопок.

update\_contents(self) – обновление окна.

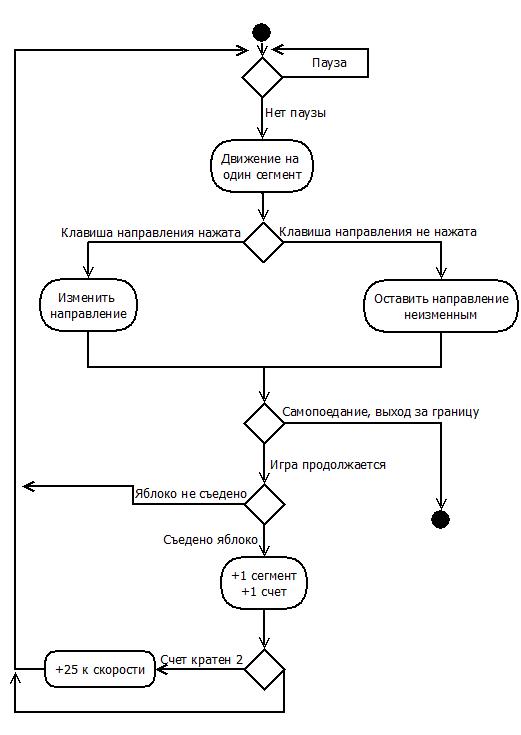
draw\_bg(self) – создание заднего фона.

draw\_map\_contents(self) – для каждого элемента матрицы, вызывается метод draw\_grid()

draw\_grid(self) – задается цвет сегмента карты и рисуется в окне.

draw\_score(self) – отображение счета и лучшего счета.

Activity diagram:



**3.Примеры кода с описанием**

Сохранение игры:

**def** save(self):  
 change = 0  
 **if** self.nonpause: #постановка паузы, если ее не было  
 self.nonpause = **False** change = 1 #переменная, позволяющая после сохранения вернуть игру в исходное состояние (пауза-не пауза)  
 **with** open(**"D:\RGZ-UltraSnake\_final\RGZ-UltraSnake\save.txt"**, **"w"**) **as** f: #открытие файла для записи  
 f.write(str(self.snake.direc\_next.value)) #запись текущего направления  
 f.write(**"\n"**)  
 f.write(str(self.score)) #запись текущего счета  
 f.write(**"\n"**)  
 f.write(str(self.conf.curr\_snake\_speed)) #запись текущей скорости  
 f.write(**"\n"**)  
 **for** i **in** range(self.snake.len()): #запись построчно и через пробел координат сегментов змейки  
 f.write(str(self.snake.bodies[i].x))  
 f.write(**" "**)  
 f.write(str(self.snake.bodies[i].y))  
 f.write(**"\n"**)  
 **if** change:  
 self.nonpause = **True** #возвращение игры в исходное состояние

Загрузка игры:

Game

**def** load(self):  
 change = 0  
 **if** self.nonpause:  
 self.nonpause = **False** change = 1  
 **with** open(**"D:\RGZ-UltraSnake\_final\RGZ-UltraSnake\save.txt"**, **"r"**) **as** f: #открытие файла для чтения

apple\_pos = f.readline().split(**"\n"**)[0].split()  
 self.map.apple = Pos(int(apple\_pos[0]), int(apple\_pos[1]))  
 temp = int(f.readline().split(**"\n"**)[0]) #прочитать данные с помощью readline().split(“\n”) – получим список из 2 элементов: направления и пустой строки. Берём первый элемент и конвертируем строку в int  
 **if** temp == 1:  
 load\_direc = Direc.LEFT  
 **elif** temp == 2:  
 load\_direc = Direc.UP  
 **elif** temp == 3:  
 load\_direc = Direc.RIGHT  
 **else**:  
 load\_direc = Direc.DOWN  
 self.score = int(f.readline().split(**"\n"**)[0]) #счет, тоже самое с направлением  
 self.conf.curr\_snake\_speed = int(f.readline().split(**"\n"**)[0]) #скорость  
 data = f.read().splitlines() #считываем построчно все остальное, а именно координаты сегментов змеи  
 load\_bodies = []  
 **for** i **in** range(len(data)):  
 temp = data[i].split() #делим каждую строку на две части, в качестве разделителя пробел  
 load\_bodies.append(Pos(int(temp[0]), int(temp[1]))) #элемент с индексом 0 – это координата x, с индексом 1 – y  
 load\_types = [PointType.HEAD] + [PointType.BODY] \* (len(data)-1) #количество типов Body, Head в зависимости от длины полученного списка с координатами  
 self.snake.load(load\_bodies, load\_types, load\_direc) #вызов метода класса Snake с соответствующими параметрами

**if** change:  
 self.nonpause = **True**

Snake

**def** load(self, load\_bodies, load\_types, load\_direc):  
 self.bodies = deque(load\_bodies) # замена существующей очереди сегментов змеи  
 self.direc\_next = load\_direc # смена направления  
 self.map.reset(**True**) # запуск метода map.reset с параметром True, что означает, что яблоко будет также загружено  
 **for** i, pos **in** enumerate(self.bodies): # присваивание соответствующих типов  
 self.map.point(pos).type = load\_types[i-1]

Пауза:

При нажатии на «пробел»:

**def** \_toggle\_pause(self):  
 **if** self.nonpause:  
 self.nonpause = **False  
 else**:  
 self.nonpause = **True**

Затем в Gui в loop функции:

**def** loop():  
 **if** callable(game\_loop):  
 game\_loop()  
 **if** self.game.nonpause:  
 ...действия...

**else**:  
 self.after(self.game.conf.curr\_snake\_speed, loop) # просто вызов метода after без действий. Что и дает паузу

Перерисовка содержимого окна:

Каждый раз при вызове loop функции выполняются методы:

**def** update\_contents(self):  
 self.canvas.delete(**"deleted"**) # удаляются элементы с тегом “deleted”  
 self.draw\_bg()  
 self.draw\_map\_contents()  
 self.draw\_score()  
  
**def** draw\_bg(self):  
 self.canvas.create\_rectangle(0, 0, self.game.conf.map\_width, self.game.conf.map\_height, fill=self.game.conf.color\_bg,  
 outline=**''**, tag=**"deleted"**)  
  
**def** draw\_map\_contents(self): # указывается размер сегмента и тип для метода draw\_grid   
 **for** i **in** range(self.game.map.num\_rows - 1):  
 **for** j **in** range(self.game.map.num\_cols - 1):  
 self.draw\_grid(j \* self.game.conf.snake\_size, i \* self.game.conf.snake\_size, self.game.map.point(Pos(i + 1, j + 1)).type)  
  
**def** draw\_grid(self, x, y, t): # рисование «точки»  
 color = 0  
 **if** t == PointType.HEAD: # цвет в зависимости от типа точки  
 color = self.game.conf.color\_snake  
 **if** t == PointType.BODY:  
 color = self.game.conf.color\_snake  
 **if** t == PointType.FOOD:  
 color = self.game.conf.color\_food  
 **if** t == PointType.WALL:  
 color = self.game.conf.color\_wall  
 **if** color == self.game.conf.color\_food:  
 self.canvas.create\_oval(x, y, x + self.game.conf.snake\_size, y + self.game.conf.snake\_size, fill=color,\  
 tag=**"deleted"**)  
 **elif** color != 0:  
 self.canvas.create\_rectangle(x, y, x + self.game.conf.snake\_size, y + self.game.conf.snake\_size,\  
 fill=color, tag=**"deleted"**)  
  
**def** draw\_score(self): # отображение счета  
 self.canvas.create\_text(700, 50, text=**"Your score:"** + str(self.game.score), fill=**"#104E8B"**, font=**"Arial 17"**) # текущий счет  
 **if not** self.game.new\_record **or** self.game.conf.curr\_snake\_speed == self.game.conf.snake\_speed-25: # лучший счет  
 self.canvas.create\_text(700, 100, text=**"Your best score:"** + str(self.game.read\_best\_score()),  
 fill=**"#104E8B"**, font=**"Arial 17"**, tag=**"deleted"**)  
 **else**: # текущий счет больше лучшего  
 self.canvas.create\_text(700, 100, text=**"New best score!"**,  
 fill=**"#104E8B"**, font=**"Arial 17"**, tag=**"deleted"**)

**4.Руководство пользователя**

Игра «Snake» предназначена для домашнего использования. Поможет вам приятно скоротать время. Для старта необходимо запустить SnakeGame.py. Управление осуществляется посредством клавиш ‘W’, ‘A’, ‘S’, ‘D’ – смена направления. ‘Space’ – пауза, ‘R’ – рестарт, ‘Esc’ **–** выход из игры. Кнопки справа на экране ‘Save’, ‘Load’ – сохранение и загрузка соответственно. При надобности данные сохранения лежат в файле ‘save.txt’. При столкновении со стенами или с телом змеи вы проигрываете. Если съесть яблоко, то змея вырастет и счет увеличится на 1. При наборе кратного 2 количества очков, скорость змейки увеличивается. Цель игры – набрать максимальное количество очков. Приятной игры!

**5.Код программы помодульно**

**SnakeGame:**

**from** Snake.Game **import** Game, Conf  
**from** Snake.Gui **import** Gui  
  
conf = Conf()  
Gui.start(Game(conf))

**Map:**

**from** enum **import** Enum  
**import** random  
  
  
**class** Map:  
 **def** \_\_init\_\_(self, num\_rows, num\_cols):  
 self.num\_rows = num\_rows  
 self.num\_cols = num\_cols  
 self.content = [[Point() **for** \_ **in** range(num\_cols)] **for** \_ **in** range(num\_rows)]  
 self.reset()  
  
 **def** reset(self, load=**False**):  
  
 **for** i **in** range(self.num\_rows):  
 **for** j **in** range(self.num\_cols):  
 self.content[i][j].type = PointType.EMPTY  
 **if** load:  
 self.point(self.apple).type = PointType.FOOD  
 **for** j **in** range(self.num\_cols):  
 self.content[0][j].type = PointType.WALL  
 self.content[int(self.num\_rows)-2][j].type = PointType.WALL  
 **for** i **in** range(self.num\_rows):  
 self.content[i][0].type = PointType.WALL  
 self.content[i][int(self.num\_cols)-2].type = PointType.WALL  
  
 **def** num\_rows(self):  
 **return** self.num\_rows  
  
 **def** num\_cols(self):  
 **return** self.num\_cols  
  
 **def** point(self, pos):  
 **return** self.content[pos.x-1][pos.y-1]  
  
 **def** create\_apple(self):  
 **for** i **in** range(self.num\_rows - 2):  
 **for** j **in** range(self.num\_cols - 2):  
 **if** self.point(Pos(i + 1, j + 1)).type == PointType.FOOD:  
 self.apple = Pos(i+1, j+1)  
 **return False** flag = **True  
 while** flag:  
 a = random.randint(2, self.num\_cols - 2)  
 b = random.randint(2, self.num\_rows - 2)  
 **if** self.point(Pos(a, b)).type != PointType.EMPTY:  
 **continue  
 else**:  
 self.point(Pos(a, b)).type = PointType.FOOD  
 self.apple = Pos(a, b)  
 flag = **False  
 return True  
  
 def** is\_empty(self, pos):  
 **return** (self.point(pos).type == PointType.EMPTY **or** \  
 self.point(pos).type == PointType.FOOD)  
  
  
**class** PointType(Enum):  
 EMPTY = 0  
 FOOD = 2  
 HEAD = 100  
 BODY = 200  
 WALL = 666  
  
  
**class** Point:  
 **def** \_\_init\_\_(self):  
 self.\_type = PointType.EMPTY  
  
 **def** type(self):  
 **return** self.\_type  
  
 **def** type(self, val):  
 self.\_type = val  
  
  
**class** Direc(Enum):  
 NONE = 0  
 LEFT = 1  
 UP = 2  
 RIGHT = 3  
 DOWN = 4  
  
  
**class** Pos:  
 **def** \_\_init\_\_(self, x=0, y=0):  
 self.x = x  
 self.y = y  
  
 **def** adj(self, direc):  
 **if** direc == Direc.LEFT:  
 **return** Pos(self.x, self.y - 1)  
 **elif** direc == Direc.RIGHT:  
 **return** Pos(self.x, self.y + 1)  
 **elif** direc == Direc.UP:  
 **return** Pos(self.x - 1, self.y)  
 **elif** direc == Direc.DOWN:  
 **return** Pos(self.x + 1, self.y)  
 **else**:  
 **return None**

**Snake:**

**from** collections **import** deque  
**from** Snake.Map **import** PointType  
  
  
**class** Snake:  
 **def** \_\_init\_\_(self, map, init\_direc, init\_bodies, init\_types):  
 self.map = map  
 self.init\_direc = init\_direc  
 self.init\_bodies = init\_bodies  
 self.init\_types = init\_types  
 self.reset(**False**)  
 self.dead = **False  
  
 def** reset(self, reset\_map=**True**):  
 self.dead = **False** self.direc = self.init\_direc  
 self.direc\_next = self.direc  
 self.bodies = deque(self.init\_bodies)  
  
 **if** reset\_map:  
 self.map.reset()  
  
 **for** i, pos **in** enumerate(self.bodies):  
 self.map.point(pos).type = self.init\_types[i]  
 **return** 0  
  
 **def** load(self, load\_bodies, load\_types, load\_direc):  
 self.bodies = deque(load\_bodies)  
 self.direc\_next = load\_direc  
 self.map.reset(**True**)  
 **for** i, pos **in** enumerate(self.bodies):  
 self.map.point(pos).type = load\_types[i-1]  
  
 **def** bodies(self):  
 **return** self.bodies  
  
 **def** len(self):  
 **return** len(self.bodies)  
  
 **def** head(self):  
 **if not** self.bodies:  
 **return None  
 return** self.bodies[0]  
  
 **def** tail(self):  
 **if not** self.bodies:  
 **return None  
 return** self.bodies[-1]  
  
 **def** move(self, new\_direc=**None**):  
 **if** new\_direc **is not None**:  
 self.direc\_next = new\_direc  
 new\_head = self.head().adj(self.direc\_next)  
 **if** self.map.is\_empty(new\_head):  
 self.map.point(self.head()).type = PointType.BODY  
 self.bodies.appendleft(new\_head)  
 self.map.point(new\_head).type = PointType.HEAD  
 self.\_rm\_tail()  
 **else**:  
 self.dead = **True  
  
 def** \_rm\_tail(self):  
 self.map.point(self.tail()).type = PointType.EMPTY  
 self.bodies.pop()  
  
 **def** add\_body(self):  
 new\_body = self.tail()  
 self.bodies.append(new\_body)  
 self.map.point(new\_body).type = PointType.BODY

**Game:**

**from** Snake.Snake **import** Snake  
**from** Snake.Map **import** Map, PointType, Direc, Pos  
**import** sys  
  
  
**class** Conf:  
 **def** \_\_init\_\_(self):  
 self.name = **'Ultimate Snake'** self.snake\_speed = 250  
 self.curr\_snake\_speed = self.snake\_speed  
 self.snake\_size = 20  
  
 self.map\_width = 800  
 self.map\_height = 600  
 self.map\_cols = 31  
 self.map\_rows = 31  
  
 self.color\_bg = **'#252624'** self.color\_snake = **'#104E8B'** self.color\_food = **'#EE0000'** self.color\_wall = **'#754600'** self.init\_direc = Direc.RIGHT  
 self.init\_bodies = [Pos(3, 6), Pos(3, 5), Pos(3, 4)]  
 self.init\_types = [PointType.HEAD] + [PointType.BODY] \* 2  
  
  
**class** Game:  
 **def** \_\_init\_\_(self, conf):  
 self.conf = conf  
 self.map = Map(conf.map\_rows, conf.map\_cols)  
 self.snake = Snake(self.map, conf.init\_direc, conf.init\_bodies, conf.init\_types)  
 self.nonpause = **True** self.score = -1  
 self.new\_record = **False  
  
 def** main(self):  
 **if** self.nonpause:  
 self.snake.move()  
 **if** self.map.create\_apple():  
 self.snake.add\_body()  
 self.score += 1  
 self.conf.curr\_snake\_speed -= self.change\_speed()  
  
 **def** keybind(self):  
 bind\_list = (  
 (**'<w>'**, **lambda** e: self.update\_direc(Direc.UP)),  
 (**'<a>'**, **lambda** e: self.update\_direc(Direc.LEFT)),  
 (**'<s>'**, **lambda** e: self.update\_direc(Direc.DOWN)),  
 (**'<d>'**, **lambda** e: self.update\_direc(Direc.RIGHT)),  
 (**'<space>'**, **lambda** e: self.\_toggle\_pause()),  
 (**'<Escape>'**, **lambda** e: self.\_\_exit\_\_()),  
 (**'<r>'**, **lambda** e: self.game\_reset())  
 )  
 **return** bind\_list  
  
 **def** update\_direc(self, new\_direc):  
 **if** (self.snake.direc\_next.value - new\_direc.value != 2) **and** (self.snake.direc\_next.value - new\_direc.value != -2):  
 self.snake.direc\_next = new\_direc  
  
 **def** game\_reset(self):  
 **if** self.new\_record:  
 **with** open(**"D:\RGZ-UltraSnake\_final\RGZ-UltraSnake\\record.txt"**, **"w"**) **as** f:  
 f.write(str(self.score))  
 self.new\_record = **False** self.score = -1  
 self.conf.curr\_snake\_speed = self.conf.snake\_speed  
 self.snake.reset()  
  
 **def** \_toggle\_pause(self):  
 **if** self.nonpause:  
 self.nonpause = **False  
 else**:  
 self.nonpause = **True  
  
 def** read\_best\_score(self):  
 **with** open(**"D:\RGZ-UltraSnake\_final\RGZ-UltraSnake\\record.txt"**, **"r"**) **as** f:  
 curr\_best\_score = int(f.read())  
 print(curr\_best\_score)  
 **if** self.score > curr\_best\_score:  
 self.new\_record = **True  
 return** curr\_best\_score  
  
 **def** change\_speed(self):  
 **if** self.score % 2 == 0 **and** self.conf.curr\_snake\_speed > 25:  
 **return** 25  
 **else**:  
 **return** 0  
  
 **def** \_\_exit\_\_(self):  
 **if** self.new\_record:  
 **with** open(**"D:\RGZ-UltraSnake\_final\RGZ-UltraSnake\\record.txt"**, **"w"**) **as** f:  
 f.write(str(self.score))  
 sys.exit()  
  
 **def** save(self):  
 change = 0  
 **if** self.nonpause:  
 self.nonpause = **False** change = 1  
 **with** open(**"D:\RGZ-UltraSnake\_final\RGZ-UltraSnake\save.txt"**, **"w"**) **as** f:  
 f.write(str(self.map.apple.x))  
 f.write(**" "**)  
 f.write(str(self.map.apple.y))  
 f.write(**"\n"**)  
 f.write(str(self.snake.direc\_next.value))  
 f.write(**"\n"**)  
 f.write(str(self.score))  
 f.write(**"\n"**)  
 f.write(str(self.conf.curr\_snake\_speed))  
 f.write(**"\n"**)  
 **for** i **in** range(self.snake.len()):  
 f.write(str(self.snake.bodies[i].x))  
 f.write(**" "**)  
 f.write(str(self.snake.bodies[i].y))  
 f.write(**"\n"**)  
 **if** change:  
 self.nonpause = **True  
   
 def** load(self):  
 change = 0  
 **if** self.nonpause:  
 self.nonpause = **False** change = 1  
 **with** open(**"D:\RGZ-UltraSnake\_final\RGZ-UltraSnake\save.txt"**, **"r"**) **as** f:  
 apple\_pos = f.readline().split(**"\n"**)[0].split()  
 self.map.apple = Pos(int(apple\_pos[0]), int(apple\_pos[1]))  
 temp = int(f.readline().split(**"\n"**)[0])  
 **if** temp == 1:  
 load\_direc = Direc.LEFT  
 **elif** temp == 2:  
 load\_direc = Direc.UP  
 **elif** temp == 3:  
 load\_direc = Direc.RIGHT  
 **else**:  
 load\_direc = Direc.DOWN  
 self.score = int(f.readline().split(**"\n"**)[0])  
 self.conf.curr\_snake\_speed = int(f.readline().split(**"\n"**)[0])  
 data = f.read().splitlines()  
 load\_bodies = []  
 **for** i **in** range(len(data)):  
 temp = data[i].split()  
 load\_bodies.append(Pos(int(temp[0]), int(temp[1])))  
 load\_types = [PointType.HEAD] + [PointType.BODY] \* (len(data)-1)  
 self.snake.load(load\_bodies, load\_types, load\_direc)  
 **if** change:  
 self.nonpause = **True**

**Gui:**

**import** tkinter  
**from** Snake.Map **import** PointType, Pos  
  
  
**class** Gui(tkinter.Tk):  
 **def** \_\_init\_\_(self, title, game=**None**):  
 super().\_\_init\_\_()  
 super().title(title)  
 super().resizable(width=**False**, height=**False**)  
 super().configure(background=game.conf.color\_bg)  
  
 **if** game **is not None**:  
 self.game = game  
 self.snake = game.snake  
  
 self.canvas = tkinter.Canvas(  
 self,  
 bg=self.game.conf.color\_bg,  
 width=self.game.conf.map\_width,  
 height=self.game.conf.map\_height,  
 highlightthickness=0  
 )  
  
 self.init\_widgets()  
 self.init\_keybindings(self.game.keybind())  
  
 @staticmethod  
 **def** start(game):  
 window = Gui(game.conf.name, game)  
 window.show(game.main)  
  
 **def** show(self, game\_loop=**None**):  
 **def** loop():  
 **if** callable(game\_loop):  
 game\_loop()  
 **if** self.game.nonpause:  
 **if not** self.snake.dead:  
 self.update\_contents()  
 self.after(self.game.conf.curr\_snake\_speed, loop)  
 **else**:  
 self.canvas.create\_text(300, 300, text=**"You lost"**, fill=**"green"**, font=**"Arial 40"**)  
 self.after(self.game.conf.curr\_snake\_speed, loop)  
 **else**:  
 self.after(self.game.conf.curr\_snake\_speed, loop)  
  
 self.after(100, loop)  
 self.mainloop()  
  
 **def** init\_keybindings(self, keybindings):  
 **if** keybindings:  
 **for** kb **in** keybindings:  
 self.bind(kb[0], kb[1])  
  
 **def** init\_widgets(self):  
 self.canvas.pack(side=tkinter.LEFT)  
 button\_save = tkinter.Button(self.canvas, width=12, text=**"Save"**, command=self.game.save)  
 button\_save.pack()  
 button\_load = tkinter.Button(self.canvas, width=12, text=**"Load"**, command=self.game.load)  
 button\_load.pack()  
 self.canvas.create\_window((700, 300), window=button\_save)  
 self.canvas.create\_window((700, 330), window=button\_load)  
  
 **def** update\_contents(self):  
 self.canvas.delete(**"deleted"**)  
 self.draw\_bg()  
 self.draw\_map\_contents()  
 self.draw\_score()  
  
 **def** draw\_bg(self):  
 self.canvas.create\_rectangle(0, 0, self.game.conf.map\_width, self.game.conf.map\_height, fill=self.game.conf.color\_bg,  
 outline=**''**, tag=**"deleted"**)  
  
 **def** draw\_map\_contents(self):  
 **for** i **in** range(self.game.map.num\_rows - 1):  
 **for** j **in** range(self.game.map.num\_cols - 1):  
 self.draw\_grid(j \* self.game.conf.snake\_size, i \* self.game.conf.snake\_size,  
 self.game.map.point(Pos(i + 1, j + 1)).type)  
  
 **def** draw\_grid(self, x, y, t):  
 color = 0  
 **if** t == PointType.HEAD:  
 color = self.game.conf.color\_snake  
 **if** t == PointType.BODY:  
 color = self.game.conf.color\_snake  
 **if** t == PointType.FOOD:  
 color = self.game.conf.color\_food  
 **if** t == PointType.WALL:  
 color = self.game.conf.color\_wall  
 **if** color == self.game.conf.color\_food:  
 self.canvas.create\_oval(x, y, x + self.game.conf.snake\_size, y + self.game.conf.snake\_size, fill=color,\  
 tag=**"deleted"**)  
 **elif** color != 0:  
 self.canvas.create\_rectangle(x, y, x + self.game.conf.snake\_size, y + self.game.conf.snake\_size,\  
 fill=color, tag=**"deleted"**)  
  
 **def** draw\_score(self):  
 self.canvas.create\_text(700, 50, text=**"Your score:"** + str(self.game.score), fill=**"#104E8B"**, font=**"Arial 17"**)  
 **if not** self.game.new\_record **or** self.game.conf.curr\_snake\_speed == self.game.conf.snake\_speed-25:  
 self.canvas.create\_text(700, 100, text=**"Your best score:"** + str(self.game.read\_best\_score()),  
 fill=**"#104E8B"**, font=**"Arial 17"**, tag=**"deleted"**)  
 **else**:  
 self.canvas.create\_text(700, 100, text=**"New best score!"**,  
 fill=**"#104E8B"**, font=**"Arial 17"**, tag=**"deleted"**)

**Вывод**

Python – очень мощный язык программирования, предоставляющий огромный функционал. В данной программе продемонстрированы базовые конструкции языка и примеры их использования. Освоив базовые конструкции языка, мы можем сказать, что его синтаксис удобнее и дружелюбнее к новичкам в сравнение с языком Си. Однако он работает медленнее, поэтому писать большой проект полностью на языке Python нецелесообразно.