МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ имени М.В.-Ломоносова

Факультет вычислительной математики и кибернетики Кафедра алгоритмических языков

Отчет о выполнении задания практикума

Сетевая игра "Танки"

Баев Тимур, 425 группа
Комиссаров Андрей, 424 группа

Москва, 2018

Оглавление

Постановка задачи

Пользователь запускает программу и выбирает быть сервером или клиентом.
В первом случае пользователь пишет только порт, во втором- IP и порт.

Как только пара пользователей найдена, начинается раунд.

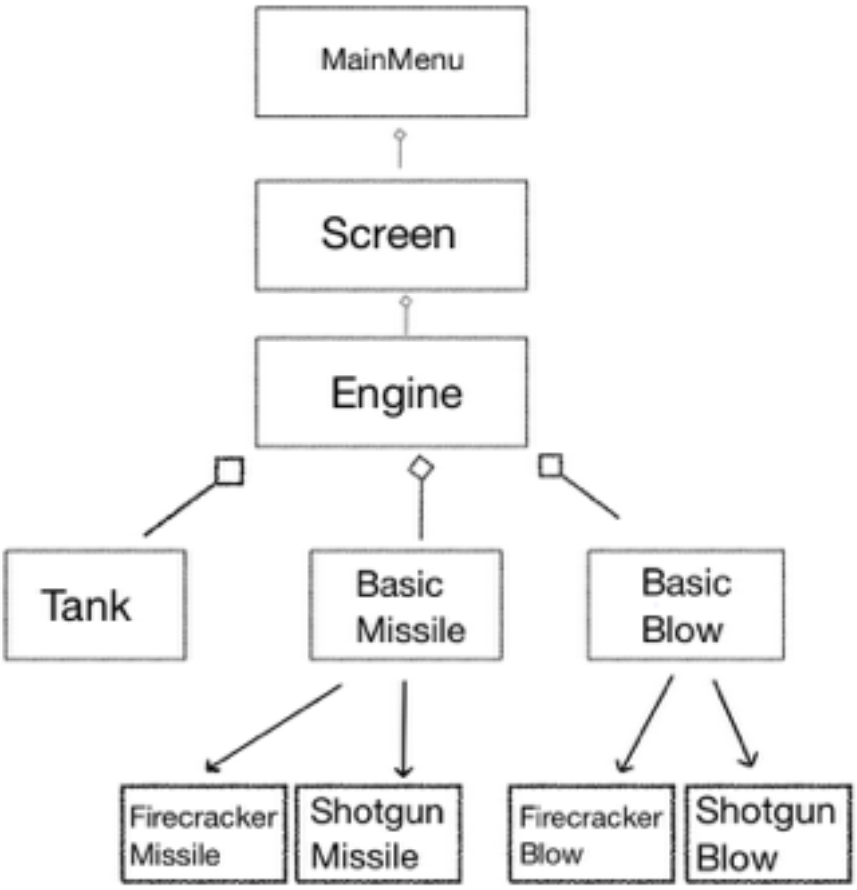
Игроки управляют своими танками в двумерном пространстве.

Цель каждого - набрать побольше баллов за попадания в чужой танк.
В арсенале каждого есть несколько различных снарядов.

Раунд заканчивается, как только набран максимум баллов.
Победитель выбирается по количеству баллов.

Далее игрокам предлагается сыграть еще один раунд.

Диаграмма основных классов



Спецификации интерфейса

Текстовая спецификация интерфейса основных классов системы

```
class Basic_Blow: # Скелет любого взрыва
        def __init__(self,engine,x,y)
        def get_name(self) # return name of this blow
        def getXY(self) # return position for this object
        def next(self,_timer=0) # move missle for next dx,dy
        def done(self) # return True if method "next" is over
        def reroze(self,_timer=0) # destroy itself and create extra blow or missle objects
in case of need
        def draw(self) # return dict with info how to draw this object

class Basic_Missile:
        def __init__(self,engine,x,y,power,angle)
        def get_name(self) # return name of this misslie
        def getXY(self) # return position for this object
        def next(self,_timer=0) # move missle for next dx,dy
        def done(self) # return True if method "next" is over or False
        def reroze(self,_timer=0) # destroy itself and create extra blow or missle objects
in case of need
        def draw(self) # return dict with info how to draw this object


class Basic_Tank:
        def __init__(self,engine,x,y,power=50,angle=None,weapons=[Basic_Missile,Firecrack-
er_Missile,Shotgun_Missile])
        def get_current_weapon_name(self) # return current weapon name
        def get_power(self) # return current tank's power
        def near(self,X,Y) # returns True if current tank is near point(X,Y)
        def get_angle(self) # return current tank's angle,return new X
        def move(self) # move changes X according to move_counter
        def getXY(self) # return current position for this obj
        def draw(self) # return dict with info how to draw this object
        def promise_move(self,task) # save promise to move, task ::= left | right (str)
        def change_angle(self,task) # changes angle , task ::= add | sub (str)
        def change_power(self,task) # changes power , task ::= add | sub (str)
        def change_weapon(self,task) # changes weapon , task ::= next | prev (str)
        def fire(self) # FIRE!!!!!!!


class Connector:
        def __init__(self,mode,host,port)
        def connect(self) # connect to peer
        def success(self) # return True if there are opened socket
        def close(self)
        def __send(self,sock,msg) # low-level send msg
        def __recv(self,sock) # low-level recv msg
        def get_ping(self,sock) # recv ping signal,return True if got ping or False
        def send_ping(self,sock) # just send ping signal
        def __answer_ping(self) # answer for ping request
        def read_msg(self) # recv msg from socket,return bytes in case of success or None
if socket is closed
        def write_msg(self,msg) # send msg into socket
        def send_weight(self,weights) # send weight's to peer
        def send_tanks(self,left_x,right_x) # send tanks' coordinates to peer
        def ping(self) # just ping peer


class Controller:
        def __init__(self)
        def run_command(self,dick)

class Engine:
        def __init__(self,canvas,draw_landscape=True)
        def f(self)
        def __find_seed(self) # return weights
```

```
        def __generate(self,weights) # return pixels
        def __draw_landscape(self) # just draw landscape and nothing more
        def blow_landscape(self,X,Y,R) # u know X, u can get Y, u should create explosion
funnel in (X,Y) with radius R
        def print_current(self) # print on canvas the current score
        def print_end(self) # print on canvas, that game is over and smbd won or game is
over
        def check_game(self) # return True if any user got score >= max score, if True =>
game is over
        def clean(self,elements=None) # delete all moveble objects from canvas
        def get_weights(self) # just return weights and nothing else
        def set_weights(self,weights) # just set new weights and nothing else
        def draw_landscape(self) # redraw landscape
        def single_draw(self) # single draw of all moveble objects
        def stop(self) # set stop flag
        def get_pixel(self,x) # return landscape Y for current x
        def is_ready(self) # return True if engine is ready for game
        def add_missile_or_blow(self,obj) # add new missile or blow
        def add_tank(self,tank,pos) # add's new tank
        def place_tanks(self,x1=None,x2=None) # place tanks

class MainMenu:
        def __init__(self):
        def get_result(self) # return the result
        def Radiobutton_1_click(self,x) # client radiobutton enabled
        def Radiobutton_2_click(self,x) # server radiobutton enabled
        def start_button_click(self,x) # start button pressed
        def exit_button_click(self,x) # exit button pressed
        def run(self) # run GUI

class Screen: # Screen with main picturebox
        def __init__(self,conn,left=True):
        def run(self) # main loop
        def place_tanks(self,x1=None,x2=None) # place tanks
        def fork(self) # start controller's
        def wait(self) # wait for both threads
        def check_game(self): # return True if any of threads are dead, if True => game is
over
        def stop_game(self,event) # instructions to stop the game
        def draw_picture(self) # callback from timer to draw picture
        def draw_landscape(self) # callback from timer to draw picture


class Socket_Controller(controller.Controller):
        def __init__(self,conn,engine,left=True)
        def stop(self) # set stop flag
        def loop(self) # main loop # do all that should do


class User_Controller(controller.Controller):
        def __init__(self,conn,engine,left=True)
        def command(self,dic) # command to run & send to peer
        def press(self,event) # button reactions
        def stop(self) # set stop flag
        def loop(self) # just loop
```

Инструментальные средства

Язык программирования: Python
Среда разработки: Sublime Text 3
Используемые библиотеки: tkinter, numpy, scipy, keyboard

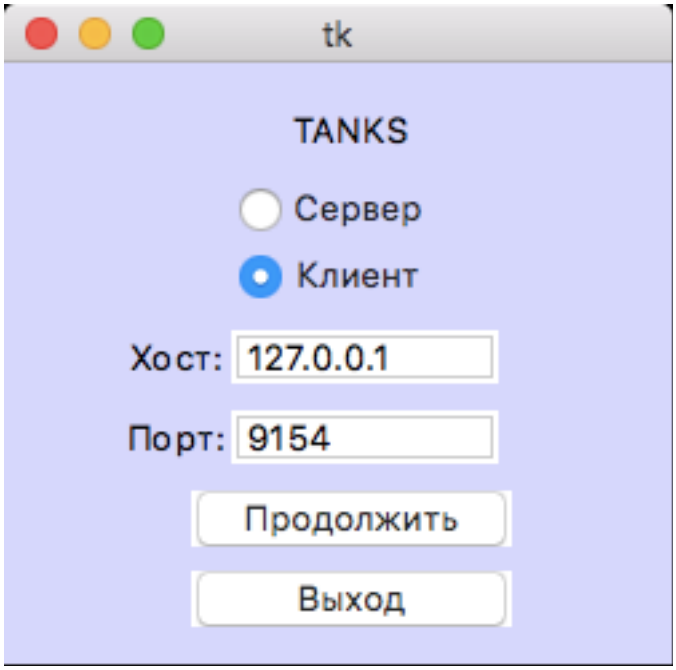
Файловая структура системы

Каждый класс лежит в своем файле

basic_blow.py
basic_missile.py
basic_tank.py
conf.py
connector.py
controller.py
engine.py
firecracker_blow.py
firecracker_blow2.py
firecracker_missile.py
firecracker_missile2.py
main.py
mainmenu.py
screen.py
shotgun_blow.py
shotgun_missile.py
shotgun_missile2.py
socket_controller.py
user_controller.py

Пользовательский интерфейс

Скриншот главного меню, где пользователь решает быть клиентом или сервером.



Игровое окно, в котором нарисованны все рабочие объекты и текущее состояние программы: