Capture the flag
A user manual

This is a user manual for our game “Capture the flag” created by Ludwig Ingestedt, Albin Ekman and Felix Strömberg

Starting the game

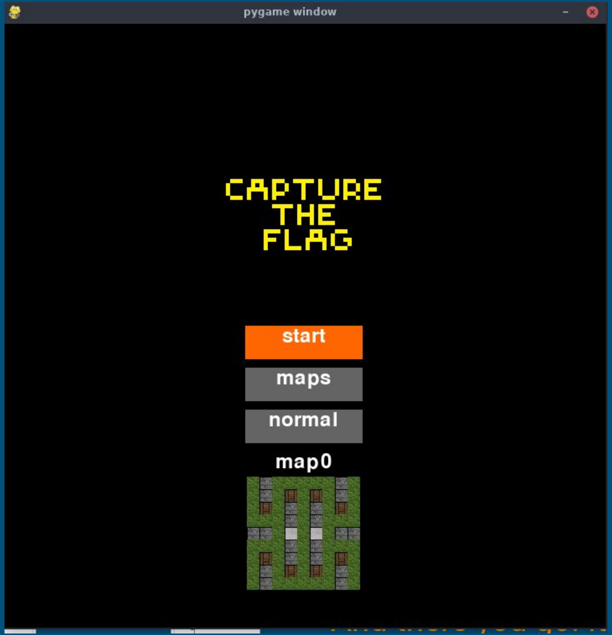
Before you can begin playing and having fun you first have to be able to launch it, So let's teach you how!

Begin by installing python3, then run the following commands in your terminal (Remember that you have to be inside the ctf folder):

- source setup.sh
- python3 ctf.py

And there you go. It's that easy!

When you launch the game it should look something like this:

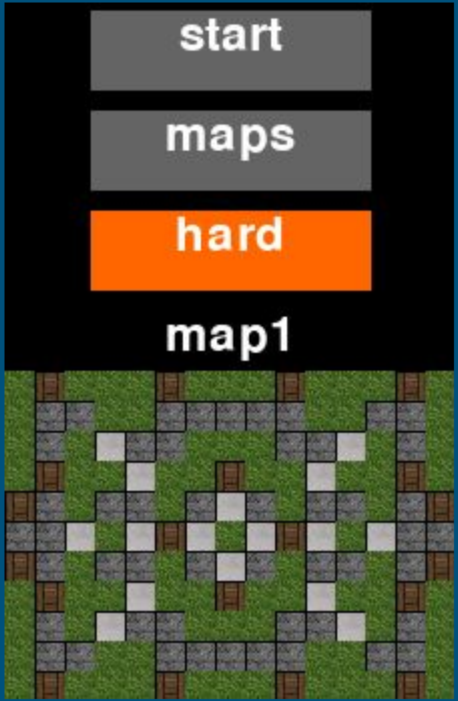


Sidenote: You are also able to add in your own map via a json file.
This is done by adding the flag —map {jsonfilename}. To learn how the format works look in the json_maps folder

How to play

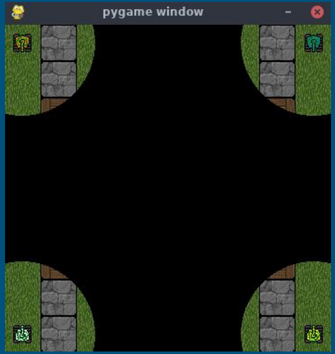
Now that the game is up and running I think it's time we teach you how to play!

The first thing you do is select both the map and the difficulty in the menu screen. This is done by using the arrow keys to move around the menu and the enter key to select



How to play

After you've pressed "start" the game should be up and running. It should look a little something like this:



Now that you're in the game, let's teach you how to win!

You move with the arrow keys and shoot with spacebar.
Your goal is to capture the flag in the middle and return to your base without being hit by enemy bullets

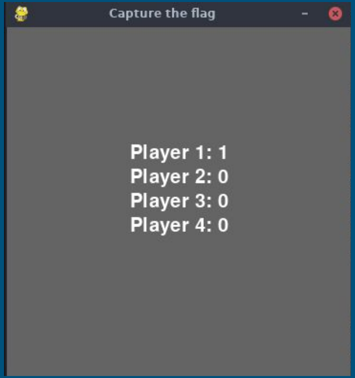
Move

Shoot

How to play

If you happen to be hit you will drop the flag(if you have it) and be sent back to your starting base. Luckily this works both ways so you're able to send back the enemy tanks too!

Once you (or someone else) successfully returns the flag they are given 1 point. Like so:



```
Player 1: 1
Player 2: 0
Player 3: 0
Player 4: 0
```

But points aren't enough for you am i right? You want to win!
There are 3 different ways to win the game and crown yourself the ultimate capture the flag champion!

1. Get 4 points
2. Have the most points after 10 rounds
3. Have the most points after 5 minutes of game time

That's all you need to know. Good luck!

Implemented features

Easy

Counting score (1pt)
Unfair Ai (1pt)

Medium

Read maps from a json file (2pts)
Welcome screen (2.5pts)
Score screen (2.5pts)
Additional win condition (3pts)
Implement fog of war (3pts)

Hard

Explaining the features

Counting score (1pt): The game keeps track of how many times each player has successfully returned the flag

Unfair Ai (1pt):
You are able to choose a difficulty setting in the menu screen. You can choose between easy, normal and hard. Depending on what difficulty you choose the game adds a multiplier to the movement speed and bullet speed of the tanks.

Read maps from a json file (2pts):
When you launch the game from the terminal you are able to import a map from a json file by adding the tag –map -{filename of json file}. The imported map is then added into the list of available maps.

Welcome screen (2.5pts):
The ctf file opens a menu screen when you run the game instead of directly putting you into the action. The menu contains three button. One for map selection, one for difficulty selection and one to start the game.

Score screen (2.5pts):
The game shows a score screen everytime someone gets a point.

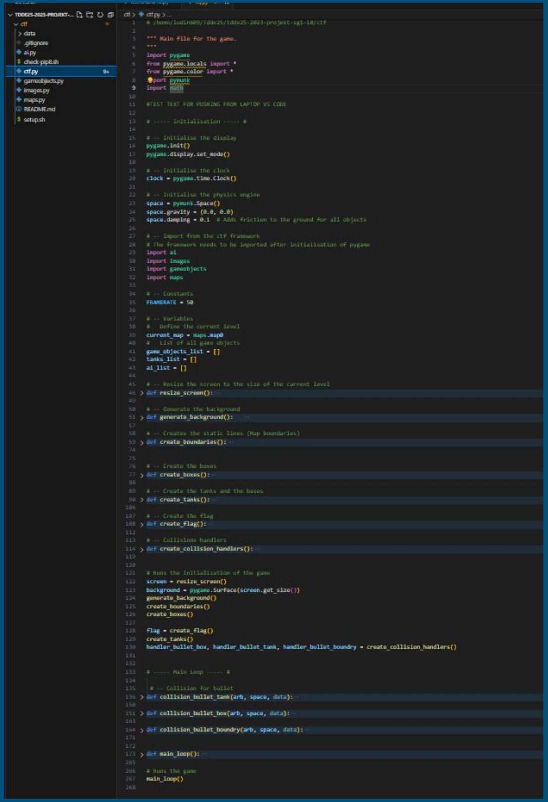
Additional win condition (3pts):
The game has 3 win conditions. Get 4 points, have the most points after 10 rounds or have the most points after 5 minutes of game time. When a winner is determined the game shows a victory screen with the winners name.

Implement fog of war (3pts):
The entire screen is blacked out except for a circle around each player so you are only able to see the map around each tank.

Explaining the source folder

The source folder is called ctf and contains the following files:

- gameobjects contains all the different classes for objects in the game e.g tank, box or bullet
- images loads and stores all the images used in the game
- maps contains the different maps available for the game
- ctf The main file of the game where the initialization and main loop of the game are located
- ai contains the ai class that decides how the bot tanks will behave



```
TDDE25-2023-PROJEKT-...
  ctf
    data
    .gitignore
    ai.py
    check-pip8.sh
    ctf.py          9+
    gameobjects.py
    images.py
    maps.py
    README.md
    setup.sh
```

ctf > ctf.py > ...

```python
# /home/ludin809/Tdde25/tdde25-2023-projekt-sg1-10/ctf

""" Main file for the game.
"""
import pygame
from pygame.locals import *
from pygame.color import *
import pymunk
import math

#TEST TEXT FOR PUSHING FROM LAPTOP VS CODE

# ----- Initialisation ----- #

# -- Initialise the display
pygame.init()
pygame.display.set_mode()

# -- Initialise the clock
clock = pygame.time.Clock()

# -- Initialise the physics engine
space = pymunk.Space()
space.gravity = (0.0, 0.0)
space.damping = 0.1  # Adds friction to the ground for all objects

# -- Import from the ctf framework
# The framework needs to be imported after initialisation of pygame
import ai
import images
import gameobjects
import maps

# -- Constants
FRAMERATE = 50

# -- Variables
#   Define the current level
current_map = maps.map0
#   List of all game objects
game_objects_list = []
tanks_list = []
ai_list = []

# -- Resize the screen to the size of the current level
def resize_screen():

# -- Generate the background
def generate_background():

# -- Creates the static lines (Map boundaries)
def create_boundaries():


# -- Create the boxes
def create_boxes():

# -- Create the tanks and the bases
def create_tanks():

# -- Create the flag
def create_flag():

# -- Collisions handlers
def create_collision_handlers():


# Runs the initialization of the game
screen = resize_screen()
background = pygame.Surface(screen.get_size())
generate_background()
create_boundaries()
create_boxes()

flag = create_flag()
create_tanks()
handler_bullet_box, handler_bullet_tank, handler_bullet_boundry = create_collision_handlers()


# ----- Main Loop ----- #

 # -- Collision for bullet
def collision_bullet_tank(arb, space, data):

def collision_bullet_box(arb, space, data):

def collision_bullet_boundry(arb, space, data):


def main_loop():

# Runs the game
main_loop()
```

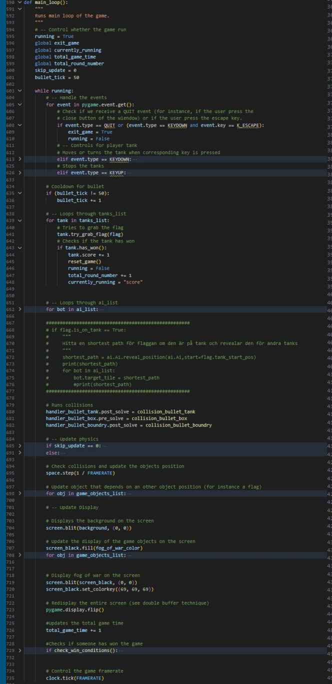
Deep dive into the files

ctf

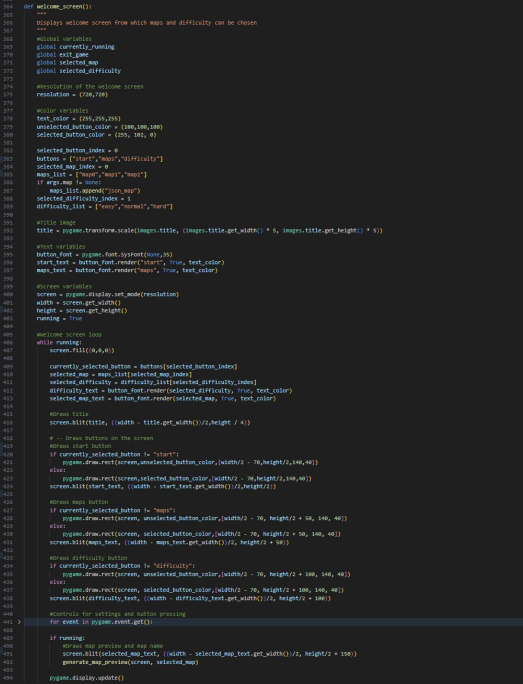
Game-loops:

Runs all of the game elements by way of functions that run loops.

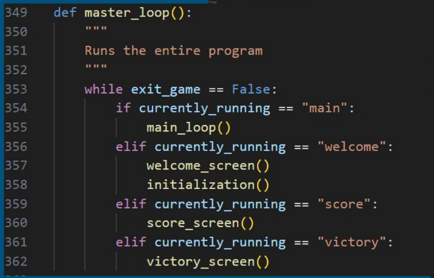
- master_loop runs the entire game by choosing which of the other loops is being run
- main_loop creates and runs the core game loop, is accessed through the welcome screen and goes into the victory screen after a win condition is achieved
- welcome_screen generates a welcome screen from which map and difficulty can be chosen
- score_screen shows current player scores between games



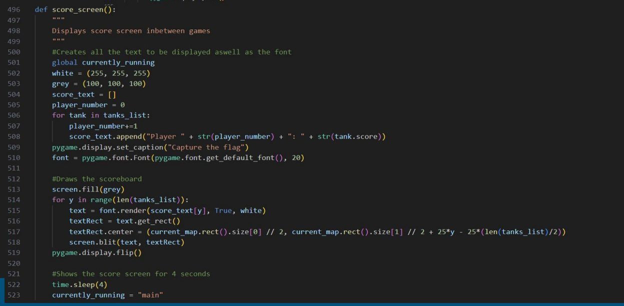
```python
def main_loop():
    """
    Runs main loop of the game.
    """
    # -- Control whether the game run
    running = True
    global exit_game
    global currently_running
    global total_game_time
    global total_round_number
    skip_update = 0
    bullet_tick = 50

    while running:
        # -- Handle the events
        for event in pygame.event.get():
            # Check if we receive a QUIT event (for instance, if the user press the
            # close button of the wiendow) or if the user press the escape key.
            if event.type == QUIT or (event.type == KEYDOWN and event.key == K_ESCAPE):
                exit_game = True
                running = False
            # -- Controls for player tank
            # Moves or turns the tank when corresponding key is pressed
            elif event.type == KEYDOWN: ···
            # Stops the tanks
            elif event.type == KEYUP: ···

        # Cooldown for bullet
        if (bullet_tick != 50):
            bullet_tick += 1

        # -- Loops through tanks_list
        for tank in tanks_list:
            # Tries to grab the flag
            tank.try_grab_flag(flag)
            # Checks if the tank has won
            if tank.has_won():
                tank.score += 1
                reset_game()
                running = False
                total_round_number += 1
                currently_running = "score"


        # -- Loops through ai_list
        for bot in ai_list: ···

        ####################################################
        # if flag.is_on_tank == True:
        #     """
        #     Hitta en shortest path för flaggan om den är på tank och revealar den för andra tanks
        #     """
        #     shortest_path = ai.Ai.reveal_position(ai.Ai,start=flag.tank_start_pos)
        #     print(shortest_path)
        #     for bot in ai_list:
        #         bot.target_tile = shortest_path
        #         #print(shortest_path)
        ####################################################

        # Runs collisions
        handler_bullet_tank.post_solve = collision_bullet_tank
        handler_bullet_box.pre_solve = collision_bullet_box
        handler_bullet_boundry.post_solve = collision_bullet_boundry

        # -- Update physics
        if skip_update == 0: ···
        else: ···

        # Check collisions and update the objects position
        space.step(1 / FRAMERATE)

        # Update object that depends on an other object position (for instance a flag)
        for obj in game_objects_list: ···

        # -- Update Display

        # Displays the background on the screen
        screen.blit(background, (0, 0))

        # Update the display of the game objects on the screen
        screen_black.fill(fog_of_war_color)
        for obj in game_objects_list: ···


        # Display fog of war on the screen
        screen.blit(screen_black, (0, 0))
        screen_black.set_colorkey((69, 69, 69))

        # Redisplay the entire screen (see double buffer technique)
        pygame.display.flip()

        #Updates the total game time
        total_game_time += 1

        #Checks if someone has won the game
        if check_win_conditions(): ···


        # Control the game framerate
        clock.tick(FRAMERATE)
```



```python
def welcome_screen():
    """
    Displays welcome screen from which maps and difficulty can be chosen
    """
    #Global variables
    global currently_running
    global exit_game
    global selected_map
    global selected_difficulty

    #Resolution of the welcome screen
    resolution = (720,720)

    #Color variables
    text_color = (255,255,255)
    unselected_button_color = (100,100,100)
    selected_button_color = (255, 102, 0)

    selected_button_index = 0
    buttons = ["start","maps","difficulty"]
    selected_map_index = 0
    maps_list = ["map0","map1","map2"]
    if args.map != None:
        maps_list.append("json_map")
    selected_difficulty_index = 1
    difficulty_list = ["easy","normal","hard"]

    #Title image
    title = pygame.transform.scale(images.title, (images.title.get_width() * 5, images.title.get_height() * 5))

    #Text variables
    button_font = pygame.font.SysFont(None,35)
    start_text = button_font.render("start", True, text_color)
    maps_text = button_font.render("maps", True, text_color)

    #Screen variables
    screen = pygame.display.set_mode(resolution)
    width = screen.get_width()
    height = screen.get_height()
    running = True

    #Welcome screen loop
    while running:
        screen.fill((0,0,0))

        currently_selected_button = buttons[selected_button_index]
        selected_map = maps_list[selected_map_index]
        selected_difficulty = difficulty_list[selected_difficulty_index]
        difficulty_text = button_font.render(selected_difficulty, True, text_color)
        selected_map_text = button_font.render(selected_map, True, text_color)

        #Draws title
        screen.blit(title, ((width - title.get_width())/2,height / 4))

        # -- Draws buttons on the screen
        #Draws start button
        if currently_selected_button != "start":
            pygame.draw.rect(screen,unselected_button_color,[width/2 - 70,height/2,140,40])
        else:
            pygame.draw.rect(screen,selected_button_color,[width/2 - 70,height/2,140,40])
        screen.blit(start_text, ((width - start_text.get_width())/2,height/2))

        #Draws maps button
        if currently_selected_button != "maps":
            pygame.draw.rect(screen, unselected_button_color,[width/2 - 70, height/2 + 50, 140, 40])
        else:
            pygame.draw.rect(screen, selected_button_color,[width/2 - 70, height/2 + 50, 140, 40])
        screen.blit(maps_text, ((width - maps_text.get_width())/2, height/2 + 50))

        #Draws difficulty button
        if currently_selected_button != "difficulty":
            pygame.draw.rect(screen, unselected_button_color,[width/2 - 70, height/2 + 100, 140, 40])
        else:
            pygame.draw.rect(screen, selected_button_color,[width/2 - 70, height/2 + 100, 140, 40])
        screen.blit(difficulty_text, ((width - difficulty_text.get_width())/2, height/2 + 100))

        #Controls for settings and button pressing
        for event in pygame.event.get(): ···

        if running:
            #Draws map preview and map name
            screen.blit(selected_map_text, ((width - selected_map_text.get_width())/2, height/2 + 150))
            generate_map_preview(screen, selected_map)

        pygame.display.update()
```



```python
def master_loop():
    """
    Runs the entire program
    """
    while exit_game == False:
        if currently_running == "main":
            main_loop()
        elif currently_running == "welcome":
            welcome_screen()
            initialization()
        elif currently_running == "score":
            score_screen()
        elif currently_running == "victory":
            victory_screen()
```



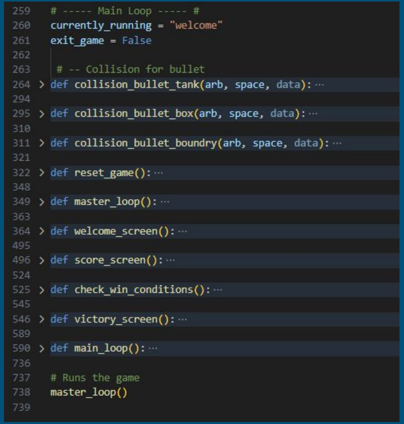
```python
def score_screen():
    """
    Displays score screen inbetween games
    """
    #Creates all the text to be displayed aswell as the font
    global currently_running
    white = (255, 255, 255)
    grey = (100, 100, 100)
    score_text = []
    player_number = 0
    for tank in tanks_list:
        player_number+=1
        score_text.append("Player " + str(player_number) + ": " + str(tank.score))
    pygame.display.set_caption("Capture the flag")
    font = pygame.font.Font(pygame.font.get_default_font(), 20)

    #Draws the scoreboard
    screen.fill(grey)
    for y in range(len(tanks_list)):
        text = font.render(score_text[y], True, white)
        textRect = text.get_rect()
        textRect.center = (current_map.rect().size[0] // 2, current_map.rect().size[1] // 2 + 25*y - 25*(len(tanks_list)/2))
        screen.blit(text, textRect)
    pygame.display.flip()

    #Shows the score screen for 4 seconds
    time.sleep(4)
    currently_running = "main"
```

Initialization and game loop:

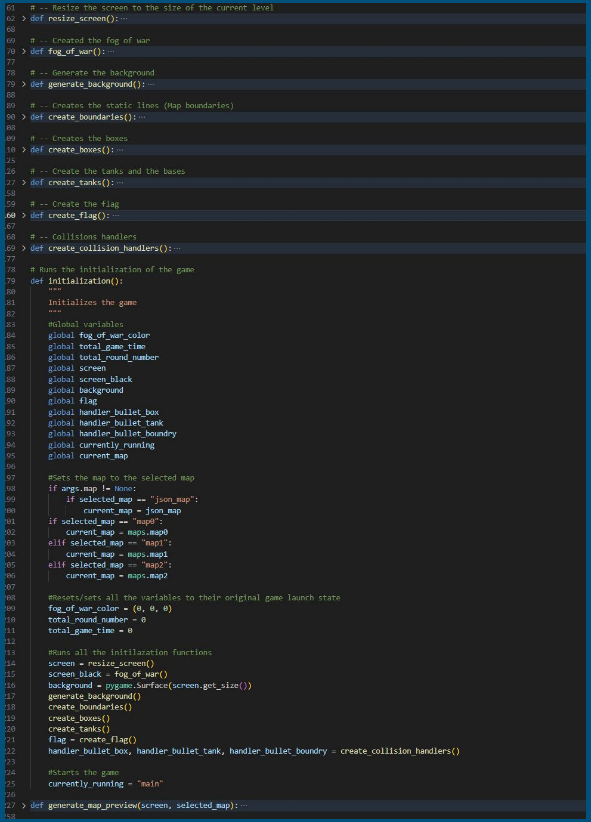
ctf

The ctf file contains two main sections housing different functions

- The initialization section contains generative functions that are called in initialization() to generate and initialize the game
- The game loop section contains functions that handle different parts of the game when it is running such as collisions and it also contains the core game loops as well. It also contains the call of the master_loop() function which runs the entire game



```
# ----- Main Loop ----- #
currently_running = "welcome"
exit_game = False

 # -- Collision for bullet
def collision_bullet_tank(arb, space, data): ···

def collision_bullet_box(arb, space, data): ···

def collision_bullet_boundry(arb, space, data): ···

def reset_game(): ···

def master_loop(): ···

def welcome_screen(): ···

def score_screen(): ···

def check_win_conditions(): ···

def victory_screen(): ···

def main_loop(): ···

# Runs the game
master_loop()
```



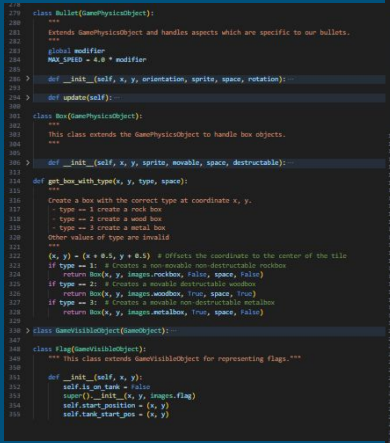
```
# -- Resize the screen to the size of the current level
def resize_screen(): ···

# -- Created the fog of war
def fog_of_war(): ···

# -- Generate the background
def generate_background(): ···

# -- Creates the static lines (Map boundaries)
def create_boundaries(): ···

# -- Creates the boxes
def create_boxes(): ···

# -- Create the tanks and the bases
def create_tanks(): ···

# -- Create the flag
def create_flag(): ···

# -- Collisions handlers
def create_collision_handlers(): ···

# Runs the initialization of the game
def initialization():
    """
    Initializes the game
    """
    #Global variables
    global fog_of_war_color
    global total_game_time
    global total_round_number
    global screen
    global screen_black
    global background
    global flag
    global handler_bullet_box
    global handler_bullet_tank
    global handler_bullet_boundry
    global currently_running
    global current_map

    #Sets the map to the selected map
    if args.map != None:
        if selected_map == "json_map":
            current_map = json_map
    if selected_map == "map0":
        current_map = maps.map0
    elif selected_map == "map1":
        current_map = maps.map1
    elif selected_map == "map2":
        current_map = maps.map2

    #Resets/sets all the variables to their original game launch state
    fog_of_war_color = (0, 0, 0)
    total_round_number = 0
    total_game_time = 0

    #Runs all the initilazation functions
    screen = resize_screen()
    screen_black = fog_of_war()
    background = pygame.Surface(screen.get_size())
    generate_background()
    create_boundaries()
    create_boxes()
    create_tanks()
    flag = create_flag()
    handler_bullet_box, handler_bullet_tank, handler_bullet_boundry = create_collision_handlers()

    #Starts the game
    currently_running = "main"

def generate_map_preview(screen, selected_map): ···
```

gameobjects

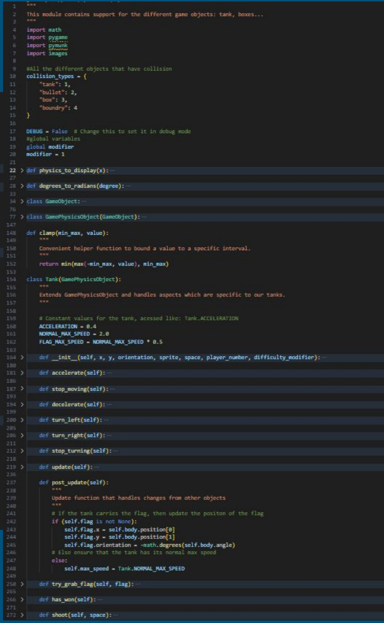
Game object classes

The gameobjects file defines classes for different objects in the game and also makes helpful functions. All the classes are derived from one of the GameObject classes for ease of handling.

- Bullet class for creating the tanks' bullets
- Tank class creates tanks and also contains functions for their movement
- Box class creates boxes
- get_box_with_type() is a function that helps easily create a box of a given type
- Flag is a class for creating a flag object that can be picked up by the tanks



```python
class Bullet(GamePhysicsObject):
    """
    Extends GamePhysicsObject and handles aspects which are specific to our bullets.
    """
    global modifier
    MAX_SPEED = 4.0 * modifier

    def __init__(self, x, y, orientation, sprite, space, rotation): ···

    def update(self): ···

class Box(GamePhysicsObject):
    """
    This class extends the GamePhysicsObject to handle box objects.
    """

    def __init__(self, x, y, sprite, movable, space, destructable): ···

def get_box_with_type(x, y, type, space):
    """
    Create a box with the correct type at coordinate x, y.
     - type == 1 create a rock box
     - type == 2 create a wood box
     - type == 3 create a metal box
    Other values of type are invalid
    """
    (x, y) = (x + 0.5, y + 0.5)  # Offsets the coordinate to the center of the tile
    if type == 1:  # Creates a non-movable non-destructable rockbox
        return Box(x, y, images.rockbox, False, space, False)
    if type == 2:  # Creates a movable destructable woodbox
        return Box(x, y, images.woodbox, True, space, True)
    if type == 3:  # Creates a movable non-destructable metalbox
        return Box(x, y, images.metalbox, True, space, False)

class GameVisibleObject(GameObject): ···

class Flag(GameVisibleObject):
    """ This class extends GameVisibleObject for representing flags."""

    def __init__(self, x, y):
        self.is_on_tank = False
        super().__init__(x, y, images.flag)
        self.start_position = (x, y)
        self.tank_start_pos = (x, y)
```



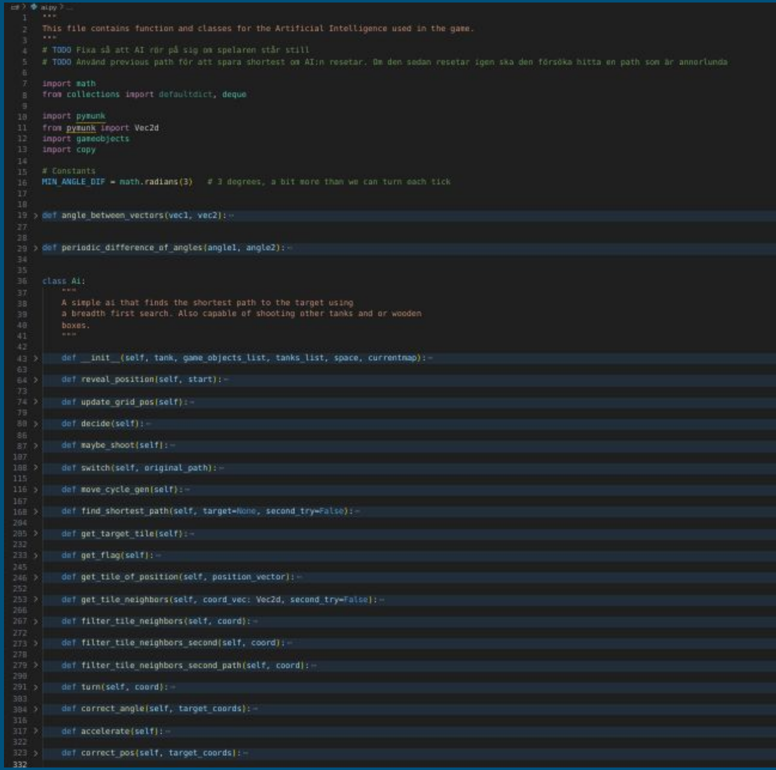
```python
"""
This module contains support for the different game objects: tank, boxes...
"""
import math
import pygame
import pymunk
import images

#All the different objects that have collision
collision_types = {
    "tank": 1,
    "bullet": 2,
    "box": 3,
    "boundry": 4
}

DEBUG = False  # Change this to set it in debug mode
#global variables
global modifier
modifier = 1

def physics_to_display(x): ···

def degrees_to_radians(degree): ···

class GameObject: ···

class GamePhysicsObject(GameObject): ···

def clamp(min_max, value):
    """
    Convenient helper function to bound a value to a specific interval.
    """
    return min(max(-min_max, value), min_max)

class Tank(GamePhysicsObject):
    """
    Extends GamePhysicsObject and handles aspects which are specific to our tanks.
    """

    # Constant values for the tank, acessed like: Tank.ACCELERATION
    ACCELERATION = 0.4
    NORMAL_MAX_SPEED = 2.0
    FLAG_MAX_SPEED = NORMAL_MAX_SPEED * 0.5

    def __init__(self, x, y, orientation, sprite, space, player_number, difficulty_modifier): ···

    def accelerate(self): ···

    def stop_moving(self): ···

    def decelerate(self): ···

    def turn_left(self): ···

    def turn_right(self): ···

    def stop_turning(self): ···

    def update(self): ···

    def post_update(self):
        """
        Update function that handles changes from other objects
        """
        # If the tank carries the flag, then update the positon of the flag
        if (self.flag is not None):
            self.flag.x = self.body.position[0]
            self.flag.y = self.body.position[1]
            self.flag.orientation = -math.degrees(self.body.angle)
        # Else ensure that the tank has its normal max speed
        else:
            self.max_speed = Tank.NORMAL_MAX_SPEED

    def try_grab_flag(self, flag): ···

    def has_won(self): ···

    def shoot(self, space): ···
```

ai

ai functions

Contains the Ai class as well as two help functions that can be used in the game.

- Ai class creates instances of bots and contains functions for controlling their tanks as well as pathfinding
- decide() runs the movement cycle and the shooting function
- move_cycle_gen() finds the shortest path to an objective and gives movement orders to follow said path
- maybe_shoot() detects and fires at shootable objects with raycasting
- find_shortest_path() uses breadth first searching to locate and return the shortest possible path to an objective (flag or base)



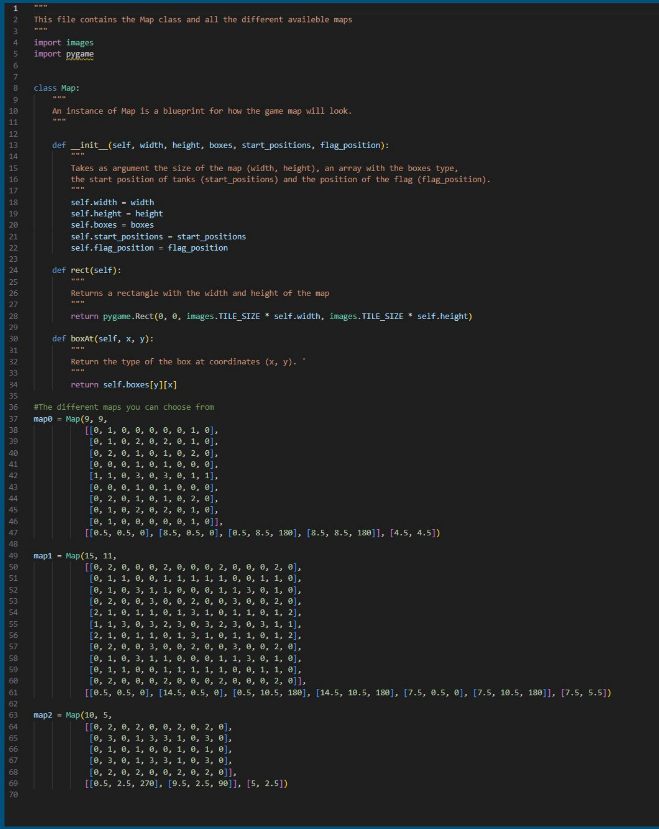
```
"""
This file contains function and classes for the Artificial Intelligence used in the game.
"""
# TODO Fixa så att AI rör på sig om spelaren står still
# TODO Använd previous path för att spara shortest om AI:n resetar. Om den sedan resetar igen ska den försöka hitta en path som är annorlunda

import math
from collections import defaultdict, deque

import pymunk
from pymunk import Vec2d
import gameobjects
import copy

# Constants
MIN_ANGLE_DIF = math.radians(3)   # 3 degrees, a bit more than we can turn each tick


def angle_between_vectors(vec1, vec2):⋯


def periodic_difference_of_angles(angle1, angle2):⋯


class Ai:
    """
    A simple ai that finds the shortest path to the target using
    a breadth first search. Also capable of shooting other tanks and or wooden
    boxes.
    """

    def __init__(self, tank, game_objects_list, tanks_list, space, currentmap):⋯

    def reveal_position(self, start):⋯

    def update_grid_pos(self):⋯

    def decide(self):⋯

    def maybe_shoot(self):⋯

    def switch(self, original_path):⋯

    def move_cycle_gen(self):⋯

    def find_shortest_path(self, target=None, second_try=False):⋯

    def get_target_tile(self):⋯

    def get_flag(self):⋯

    def get_tile_of_position(self, position_vector):⋯

    def get_tile_neighbors(self, coord_vec: Vec2d, second_try=False):⋯

    def filter_tile_neighbors(self, coord):⋯

    def filter_tile_neighbors_second(self, coord):⋯

    def filter_tile_neighbors_second_path(self, coord):⋯

    def turn(self, coord):⋯

    def correct_angle(self, target_coords):⋯

    def accelerate(self):⋯

    def correct_pos(self, target_coords):⋯
```

maps

maps functions

Contains the Map class as well as several different maps that can be used in the game. Map selection is done from the welcome screen

- Map class creates instances of different maps
- map0 is the default map with the dimensions 9 by 9 tiles featuring 4 tanks
- map1 is a map with the dimensions 15 by 11 tiles featuring 6 tanks
- map2 is a map with the dimensions 10 by 5 tiles featuring 2 tanks



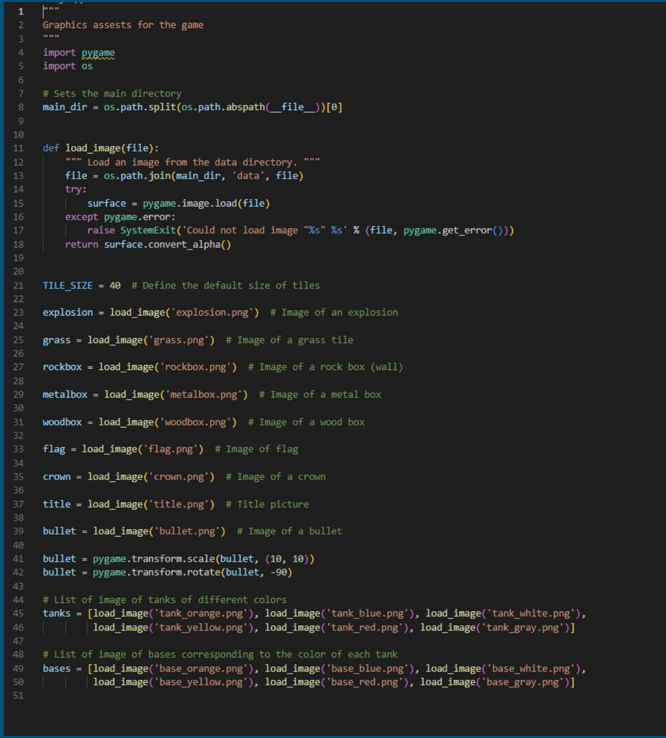
```python
"""
This file contains the Map class and all the different availeble maps
"""
import images
import pygame


class Map:
    """
    An instance of Map is a blueprint for how the game map will look.
    """

    def __init__(self, width, height, boxes, start_positions, flag_position):
        """
        Takes as argument the size of the map (width, height), an array with the boxes type,
        the start position of tanks (start_positions) and the position of the flag (flag_position).
        """
        self.width = width
        self.height = height
        self.boxes = boxes
        self.start_positions = start_positions
        self.flag_position = flag_position

    def rect(self):
        """
        Returns a rectangle with the width and height of the map
        """
        return pygame.Rect(0, 0, images.TILE_SIZE * self.width, images.TILE_SIZE * self.height)

    def boxAt(self, x, y):
        """
        Return the type of the box at coordinates (x, y). '
        """
        return self.boxes[y][x]

#The different maps you can choose from
map0 = Map(9, 9,
            [[0, 1, 0, 0, 0, 0, 0, 1, 0],
             [0, 1, 0, 2, 0, 2, 0, 1, 0],
             [0, 2, 0, 1, 0, 1, 0, 2, 0],
             [0, 0, 0, 1, 0, 1, 0, 0, 0],
             [1, 1, 0, 3, 0, 3, 0, 1, 1],
             [0, 0, 0, 1, 0, 1, 0, 0, 0],
             [0, 2, 0, 1, 0, 1, 0, 2, 0],
             [0, 1, 0, 2, 0, 2, 0, 1, 0],
             [0, 1, 0, 0, 0, 0, 0, 1, 0]],
            [[0.5, 0.5, 0], [8.5, 0.5, 0], [0.5, 8.5, 180], [8.5, 8.5, 180]], [4.5, 4.5])

map1 = Map(15, 11,
            [[0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0],
             [0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0],
             [0, 1, 0, 3, 1, 1, 0, 0, 0, 1, 1, 3, 0, 1, 0],
             [0, 2, 0, 0, 3, 0, 0, 2, 0, 0, 3, 0, 0, 2, 0],
             [2, 1, 0, 1, 1, 0, 1, 3, 1, 0, 1, 1, 0, 1, 2],
             [1, 1, 3, 0, 3, 2, 3, 0, 3, 2, 3, 0, 3, 1, 1],
             [2, 1, 0, 1, 1, 0, 1, 3, 1, 0, 1, 1, 0, 1, 2],
             [0, 2, 0, 0, 3, 0, 0, 2, 0, 0, 3, 0, 0, 2, 0],
             [0, 1, 0, 3, 1, 1, 0, 0, 0, 1, 1, 3, 0, 1, 0],
             [0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0],
             [0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0]],
            [[0.5, 0.5, 0], [14.5, 0.5, 0], [0.5, 10.5, 180], [14.5, 10.5, 180], [7.5, 0.5, 0], [7.5, 10.5, 180]], [7.5, 5.5])

map2 = Map(10, 5,
            [[0, 2, 0, 2, 0, 0, 2, 0, 2, 0],
             [0, 3, 0, 1, 3, 3, 1, 0, 3, 0],
             [0, 1, 0, 1, 0, 0, 1, 0, 1, 0],
             [0, 3, 0, 1, 3, 3, 1, 0, 3, 0],
             [0, 2, 0, 2, 0, 0, 2, 0, 2, 0]],
            [[0.5, 2.5, 270], [9.5, 2.5, 90]], [5, 2.5])
```

images

images file

Loads all the images in the game from the data folder and assigns them as objects that can be called from the ctf file

- load_image() function that takes the name of an image files and tries to load a file of the same name from the data folder
- tanks list that contains the images of all the tanks
- bases list that contains the images of all the bases/starting points



```python
"""
Graphics assests for the game
"""
import pygame
import os

# Sets the main directory
main_dir = os.path.split(os.path.abspath(__file__))[0]


def load_image(file):
    """ Load an image from the data directory. """
    file = os.path.join(main_dir, 'data', file)
    try:
        surface = pygame.image.load(file)
    except pygame.error:
        raise SystemExit('Could not load image "%s" %s' % (file, pygame.get_error()))
    return surface.convert_alpha()


TILE_SIZE = 40  # Define the default size of tiles

explosion = load_image('explosion.png')  # Image of an explosion

grass = load_image('grass.png')  # Image of a grass tile

rockbox = load_image('rockbox.png')  # Image of a rock box (wall)

metalbox = load_image('metalbox.png')  # Image of a metal box

woodbox = load_image('woodbox.png')  # Image of a wood box

flag = load_image('flag.png')  # Image of flag

crown = load_image('crown.png')  # Image of a crown

title = load_image('title.png')  # Title picture

bullet = load_image('bullet.png')  # Image of a bullet

bullet = pygame.transform.scale(bullet, (10, 10))
bullet = pygame.transform.rotate(bullet, -90)

# List of image of tanks of different colors
tanks = [load_image('tank_orange.png'), load_image('tank_blue.png'), load_image('tank_white.png'),
         load_image('tank_yellow.png'), load_image('tank_red.png'), load_image('tank_gray.png')]

# List of image of bases corresponding to the color of each tank
bases = [load_image('base_orange.png'), load_image('base_blue.png'), load_image('base_white.png'),
         load_image('base_yellow.png'), load_image('base_red.png'), load_image('base_gray.png')]
```