

ID: 211962212 ,Name: Nir Bendov

```

classDiagram
    class GameBoard {
        - board: char[][]
        - player1Tanks: Tank[]
        - player2Tanks: Tank[]
        - bulletsPositions: Shell[]
        - stepMoves: Action[]
        - gameOver: bool
        - winner: int
        - numOfTurnsSinceNoAmmo: int
        - tankDeaths: TankDeath[]
        + validateMove(Action a, PlayerId i): bool
        + addToStepMoves(Action a, PlayerId i): void
        + executeStep(): void
        + saveGameMoves(string fileName): void
        + printBoard(): void
    }
    class TankDeath {
        - position: int[2]
        - playerId: PlayerId
        - cause: string
    }
    class Tank {
        - ammoCount: int
        - moveBackwardCooldown: int
        - shootingCooldown: int
        - isMovingBackward: bool
        - finishedMovingBackward: bool
        - isTankAlive: bool
        - algo: Algorithm*
        - playerId: PlayerId
        + shoot(): Shell*
        + turn(Turn t): void
        + decreaseMoveBackwardCooldown(): void
        + decreaseShootinhCooldown(): void
        + moveForward(): void
        + moveBackward(): void
        + cancelMoveBackward(): void
        + killTank(): void
        + doNothing(): void
    }
    class Shell {
        - justFired: bool
    }
    class Pathfinder {
        + bfsPathFinder(char[][] grid, Point start, Point end, bool includeWalls): Point[]
        + updatePathEnd(Point[] path, Point newEnd, int height, int length): void
        + updatePathStart(Point[] path, Point newStart, int height, int length): void
        + isPathStraight(Point[] path, int height, int length): bool
        + isPathClear(Point[] path, char[][] grid): bool
        + calcDirection(Point[] path, int height, int length): int[2]
    }
    class Point {
        - x: int
        - y: int
    }
    class Action {
        - target: Tank*
        - type: Type
    }
    class Algorithm {
        + playerId: PlayerId
        + tank: Tank*
        + gameBoard: GameBoard*
        + decideNextActions(): Action[]
        + defaultMode(): void
        + update(): void
    }
    class AlgorithmBasic
    class AlgorithmPlayerTwo
    class BoardConstants {
        + WALL
        + DamagedWall
        + Player1Tank
        + Player2Tank
        + Height
        + Width
    }
    class Direction {
        + isbulletInPath() bool
        + Rotation() Turn
        + getDirection() array<int,2>
    }
    class Info {
        - direction: int[2]
        - location: int[2]
    }
    class Moveable {
        + move(): void
        + moveBack(): void
    }
    class Turn {
        RIGHT_90
        RIGHT_45
        LEFT_45
        LEFT_90
        RIGHT_135
        LEFT_135
        COMPLETE_180
        NONE_0
    }
    GameBoard "1" *-- "0..*" Tank
    GameBoard "1" *-- "0..*" Shell
    GameBoard "1" *-- "0..*" Algorithm
    GameBoard "1" *-- "0..*" TankDeath
    GameBoard "1" *-- "0..*" BoardConstants
    GameBoard "1" *-- "0..*" Direction
    GameBoard "1" *-- "0..*" Info
    GameBoard "1" *-- "0..*" Moveable
    GameBoard "1" *-- "0..*" Turn
    Tank "0..*" -- "0..*" Shell
    Tank "0..*" -- "0..*" Algorithm
    Tank "0..*" -- "0..*" TankDeath
    Tank "0..*" -- "0..*" BoardConstants
    Tank "0..*" -- "0..*" Direction
    Tank "0..*" -- "0..*" Info
    Tank "0..*" -- "0..*" Moveable
    Tank "0..*" -- "0..*" Turn
    Shell "0..*" -- "0..*" Algorithm
    Shell "0..*" -- "0..*" TankDeath
    Shell "0..*" -- "0..*" BoardConstants
    Shell "0..*" -- "0..*" Direction
    Shell "0..*" -- "0..*" Info
    Shell "0..*" -- "0..*" Moveable
    Shell "0..*" -- "0..*" Turn
    Algorithm "0..*" -- "0..*" Tank
    Algorithm "0..*" -- "0..*" Shell
    Algorithm "0..*" -- "0..*" TankDeath
    Algorithm "0..*" -- "0..*" BoardConstants
    Algorithm "0..*" -- "0..*" Direction
    Algorithm "0..*" -- "0..*" Info
    Algorithm "0..*" -- "0..*" Moveable
    Algorithm "0..*" -- "0..*" Turn
    AlgorithmBasic <|-- Algorithm
    AlgorithmPlayerTwo <|-- Algorithm
    TankDeath <|-- Tank
    TankDeath <|-- Shell
    TankDeath <|-- Algorithm
    TankDeath <|-- TankDeath
    TankDeath <|-- BoardConstants
    TankDeath <|-- Direction
    TankDeath <|-- Info
    TankDeath <|-- Moveable
    TankDeath <|-- Turn
    BoardConstants <|-- Tank
    BoardConstants <|-- Shell
    BoardConstants <|-- Algorithm
    BoardConstants <|-- TankDeath
    BoardConstants <|-- BoardConstants
    BoardConstants <|-- Direction
    BoardConstants <|-- Info
    BoardConstants <|-- Moveable
    BoardConstants <|-- Turn
    Direction <|-- Tank
    Direction <|-- Shell
    Direction <|-- Algorithm
    Direction <|-- TankDeath
    Direction <|-- BoardConstants
    Direction <|-- Direction
    Direction <|-- Info
    Direction <|-- Moveable
    Direction <|-- Turn
    Info <|-- Tank
    Info <|-- Shell
    Info <|-- Algorithm
    Info <|-- TankDeath
    Info <|-- BoardConstants
    Info <|-- Direction
    Info <|-- Info
    Info <|-- Moveable
    Info <|-- Turn
    Moveable <|-- Tank
    Moveable <|-- Shell
    Moveable <|-- Algorithm
    Moveable <|-- TankDeath
    Moveable <|-- BoardConstants
    Moveable <|-- Direction
    Moveable <|-- Info
    Moveable <|-- Moveable
    Moveable <|-- Turn
    Turn <|-- Tank
    Turn <|-- Shell
    Turn <|-- Algorithm
    Turn <|-- TankDeath
    Turn <|-- BoardConstants
    Turn <|-- Direction
    Turn <|-- Info
    Turn <|-- Moveable
    Turn <|-- Turn
  
```

2. High Level design:



3. Main loop explanation:

- a. Our main function only runs the main loop with no other logic implemented there.
- b. During start up with initialization we get the matrix from InputHandler. This class handles the loading.
- c. We pass the matrix and initialize the GameBoard.
- d. After this we initialize each algorithm and assign it a player, the algorithm pulls the player tanks from the GameBoard.
- e. Each turn the main asks each algorithm for his next and passes them to the GameBoard
- f. The GameBoard validates the moves and adds only valid steps.
- g. The main loop then asks the GameBoard to perform the step.
- h. The GameBoard moves the shells 2 times, after each time checks for collisions, and later performs the moves.
- i. We repeat the loop until the game is over or a step limit is reached.
- j. At the end we output the moves, the tank deaths and the winner to a txt file.

4. A few sidenotes:

- a. We handle collisions by setting up special characters in the GameBoard.
- b. All the special characters and normal are stored in BoardConstants.h
- c. Our GameBoard later loops through the board and handles each collision by its case.
- d. We could have handled each collision right as they go but this will make handling complex collisions harder.
- e. The only collision we handle as they go are “step overs” , a case where 2 game characters (shells or tanks) will move in the opposite direction and are next to each other and will hop over each other.

5. Class organization:

- a. We created a utility class called direction to help the algorithms with rotation and bullet paths.
- b. We created a class MoveAble that tanks and shells inherit from that handles the move logic to keep it in a single place.
- c. We focused on giving each class a unique purpose and direction:
 - 1) The GameBoard - handles all the game logic, from moving the pieces to handling collisions.
 - 2) The algorithms - We have 2 each implementing an interface called algorithm that handles how our main loop interacts with them
 - 3) The tank - Handles shooting, performing back moves and state (alive or dead).
 - 4) The shell - a small class to indicate a shell.
 - 5) InputHandler - handler the gameBoard loading.

6. Alternatives:

- a. We could have added a new class Called GameEngine that will handle the loop, and the main function will only initialize the file

7. The Algorithms:

a. AlgorithmPlayerTwo:

- 1) Find the shortest clear path to the enemy tank without destroying any walls.
- 2) If none exists, find the shortest path that does require breaking walls.
- 3) After each move by either tank, recompute the chosen path.
- 4) If a straight line (with or without walls) opens between you and the enemy, rotate to face them and fire.
- 5) When an incoming shell along your current heading is ≤ 3 steps away, attempt to dodge by:
 - a) Stepping forward if it removes you from its trajectory, or
 - b) Rotating to a safe angle and then stepping forward.
- 6) If no path exists at all (even with wall destruction), enter panic mode: fire continuously to empty your magazine and end the game as quickly as possible.

b. The Basic:

- 1) Will try to run away if in danger
- 2) Shoot if in site of an enemy
- 3) Turn to be able to shoot if possible
- 4) Move along the x axis closer to the enemy else.

8. Our testing methodology:

- a. Automatic testing - we created a few unit testing for special cases to check the game board performs as expected, they each give it a board and a set of moves for each player and them and see what will happen.
- b. We basically let a tank move by how we wanted it to move and so it will die and then we compare why,how it died and the winner to the expected .
- c. Running examples- We wrote and tested examples to check how the algorithm works, and looked at what he does at each step.