The phase II implementations involve programming to the API developed during the Phase I. The platform developed during Phase I is a tank battle game which involves a 2-d playfield of sizes varying from 5x5 to 50x20. Computer controlled players (tanks) can number between 2 and 10. The playfield can include a variety of obstacles with set characteristics.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Obstacle Name | Destructible | Traversable | Does Damage | Explodes w/splash damage | Shots pass through |
| Bush | Yes | Yes | No | No | No |
| Tree | Yes | Yes | No | No | No |
| Crate | Yes | Yes | No | Yes | No |
| Rock | Yes | No | Yes | No | No |
| Tank Buster | No | No | Yes | No | No |
| Water | No | No | Yes | No | Yes |

Tanks have a number of attributes which impact their ability to shoot, move, and explore the environment.

|  |  |
| --- | --- |
| Attribute Name | Description |
| Damage | Number of points of damage done by projectiles |
| Health | Number of points of damage the tank can absorb before being destroyed |
| Radar | Distance at which the tank can detect enemies and obstacles |
| AP | Action points available for each turn |
| Range | Distance projectiles move during a turn |
| Ammo | Number of shots available before requiring a reload |

At the beginning of a game, the tank program is given a number of SPECIAL points which can be allocated to increase its attributes. This can add considerable variability in the capabilities of a tank when strategically assigned.

There are three stages of solo tank testing with a number of levels in each stage. In all stages and levels, tanks will be allocated 2 SPECIAL points.

## Simple Navigation (30 points)

### Level 1 [For full points, tanks must ‘win’ 100% of these tests]

* No obstacles (other than the playfield boundaries)
* 1 or 2 immobile targets (Target)
* Tank has full visibility of the playfield

### Level 2 [Full points require 100% ‘win’ rate]

* No obstacles
* 1 or 2 Fleeing targets (Mouse)
* Tank has full visibility of the playfield

### Level 3 [Full points require 95% ‘win’ rate]

* No obstacles
* 1 immobile or 1 fleeing target
* Tank has limited visibility of the playfield

## Obstacle Avoidance (30 points)

### Level 1 [For full points, tanks must ‘win’ 100% of these tests]

* Sparse (non-damaging) obstacles
* 1 or 2 immobile targets
* Tank has full visibility of playfield

### Level 2 [For full points, tanks must ‘win’ 100% of these tests]

* Sparse (non-damaging) obstacles
* 1 or 2 fleeing targets
* Tank has full visibility of playfield

### Level 3 [For full points, tanks must ‘win’ 90% of these tests]

* Sparse (non-damaging) obstacles
* 1 immobile or 1 fleeing target
* Tank has limited visibility of playfield

## Offensive Opponent (30 points)

### Level 1 [For full points, tanks must ‘win’ 90% of these tests]

* No obstacles
* 1 or 2 Immobile targets that can shoot back
* Tank has full visibility of playfield

### Level 2 [For full points, tanks must ‘win’ 90% of these tests]

* Sparse obstacles
* 1 or 2 immobile targets that can shoot back
* Tank has full visibility of playfield

### Level 3 [For full points, tanks must ‘win’ 80% of these tests]

* Sparse obstacles
* 1 or 2 immobile targets that shoot back
* Tank has limited visibility

There are two additional stages of testing that will pit student tanks against other student tanks for bonus points.

## Head to Head (18 bonus points available)

### Level 1 [1 SPECIAL point]

* Tanks will be randomly matched against one other student tank
* Playfields will be random rectangles between 5 x 5 and 50 x 20
* No obstacles
* Visibility limited

### Level 2 [2 SPECIAL points]

* Tanks will be randomly matched against one other student tank
* Playfields will be random rectangles between 5 x 5 and 50 x 20
* Sparse obstacles (non-damaging)
* Visibility limited

### Level 3 [3 SPECIAL points]

* Randomly matched 1 v 1
* Random Sized playfields
* Medium obstacles (all types)
* Full field visibility

## Free For All (18 bonus points available)

### Level 1 [1 SPECIAL point]

* All student tanks in the same playfield. Playfields will range from 15 x 15 to 50 x 20
* No obstacles
* Limited Visibility

### Level 2 [2 SPECIAL points]

* All student tanks [15 x 15 to 50 x 20]
* Sparse obstacles (non-damaging)
* Limited Visibility

### Level 3 [4 SPECIAL points]

* All student tanks on same playfield [15 x 15 to 50 x 20]
* Medium obstacles (all types)
* Full visibility

# Scoring

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Simple Navigation | 100% | 90-99% | 80-89% | 70-79% | 60-69% | < 60% |
| Level 1 | 10 | 8 | 6 | 4 | 2 | 0 |
| Level 2 | 10 | 8 | 6 | 4 | 2 | 0 |
| Level 3 | 10 | 10 | 8 | 6 | 4 | 2 |
| Obstacle Avoidance | 100% | 90-99% | 80-89% | 70-79% | 60-69% | < 60% |
| Level 1 | 10 | 8 | 6 | 4 | 2 | 0 |
| Level 2 | 10 | 8 | 6 | 4 | 2 | 0 |
| Level 3 | 10 | 10 | 8 | 6 | 4 | 2 |
| Offensive Opponent | 100% | 90-99% | 80-89% | 70-79% | 60-69% | < 60% |
| Level 1 | 10 | 10 | 8 | 6 | 4 | 2 |
| Level 2 | 10 | 10 | 8 | 6 | 4 | 2 |
| Level 3 | 10 | 10 | 10 | 8 | 6 | 4 |

# Bonus

|  |  |  |  |
| --- | --- | --- | --- |
| Head to Head | 1st Place | 2nd Place | 3rd Place |
| Level 1 | 3 | 2 | 1 |
| Level 2 | 3 | 2 | 1 |
| Level 3 | 3 | 2 | 1 |
| Free For All | 1st Place | 2nd Place | 3rd Place |
| Level 1 | 3 | 2 | 1 |
| Level 2 | 3 | 2 | 1 |
| Level 3 | 3 | 2 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Archangel | Level 1 | Level 2 | Level 3 | Total |
| Simple Navigation | 100% (10) | 100% (10) | 100% (10) | 30 |
| Obstacle Avoidance | 100% (10) | 100% (10) | 100% (10) | 30 |
| Offensive Opponent | 100% (10) | 65% (4) | 31% (4) | 18 |
| Head To Head | 3rd Place (1) | 3rd Place (1) | 2nd Place(2) | 4 points |
| Free For All | 3rd Place (1) | 3rd Place (1) | 1st Place (3) | 5 points |
| Totals | 32 | 26 | 29 | 87 |

# Code / Repo Review (83/100)

* Your .gitlab-ci.yml file does not run any tests. It just builds the project
* You’ve combined the platform repository with your own. This means you’re duplicating all those binary files as well. It also means you’re committing those files every time the platform changes. This confuses the commit logs since now we see the maintenance team with what appear to be commits into your respository.

33 David Donahue <david.donahue@mines.sdsmt.edu>

33 Doering, William <william.doering@mines.sdsmt.edu>

21 jamckee <dj32mckee@hotmail.com>

17 Jon McKee <dj32mckee@hotmail.com>

6 Michael Theesen <michael.theesen@mines.sdsmt.edu>

6 Riley Kopp <koppril7@gmail.com>

4 Riley Kopp <riley.kopp@mines.sdsmt.edu>

4 Theesen, Michael <michael.theesen@mines.sdsmt.edu>

2 Donahue, David [david.donahue@mines.sdsmt.edu](mailto:david.donahue@mines.sdsmt.edu)

There’s a troubling absence of activity from some team members and a significant imbalance of repository activity as well in some cases. I do not see significant evidence of team or paired programming in the commit messages. No evidence that these team members contributed in other ways (i.e. writing tests, creating Wiki documentation, performing code reviews). The gitlab repo does not show evidence that the contributions were balanced.

* There is a test directory but the tests do not build and it does not appear from the CI pipelines that they were ever run in an automated fashion.
* In general, the repository shows significant activity in the last week or so of the project and little before that. This project was given 5+ weeks to complete.
* Good job setting up the Dockerfile to make development consistent. Having some documentation to describe the container you’re building and how it’s used would be helpful. Not everyone likes to read Dockerfiles.
* There are source files that are not used to build the final product. It was requested that the repository (at least the master branch) be cleaned up before final submission. If the example tanks are being used for testing they should be moved out of the root directory and set up to be built and run automatically.
* You have left editor backup files in the directory cluttering things up (e.g. docker\_run.sh~)
* Code has excessive indentation which causes line wrap and makes reading the code unpleasant.
* Archangel::attack could be replaced with a map or unordered map.
* ::spendAP is doing too much. Finds hostiles, copies map, then does some logic to decide whether to move or attack. This should be refactored and broken into logical methods.
* ::get\_danger has a bunch of magic numbers. What relationship do 1 and 5 have to one another? 7 and 3?. Use named constants or enums. The comment block indicates that 1 is UP and 5 is DOWN. Do the statements of the form firing\_arc.push\_back((temp\_y > 0) ? 1 : 5) indicate there is some preference for one direction or another? How would a reader know?
* WaveFront::genMap definitely needs some commentary. What is it doing? How is it doing it? What’s the returned waveMap represent?
* Everything in the WaveFront class is public. Why make it a class?
* Not a single comment in WaveFront or HeatMap. Formatting is all over the place. Did someone code review these classes?
* Not a great look all in all especially given you had 5+ weeks to apply solid Software Engineering and clean code principles to this very simple application.

# Phase II Presentation (No Phase II Presentation)