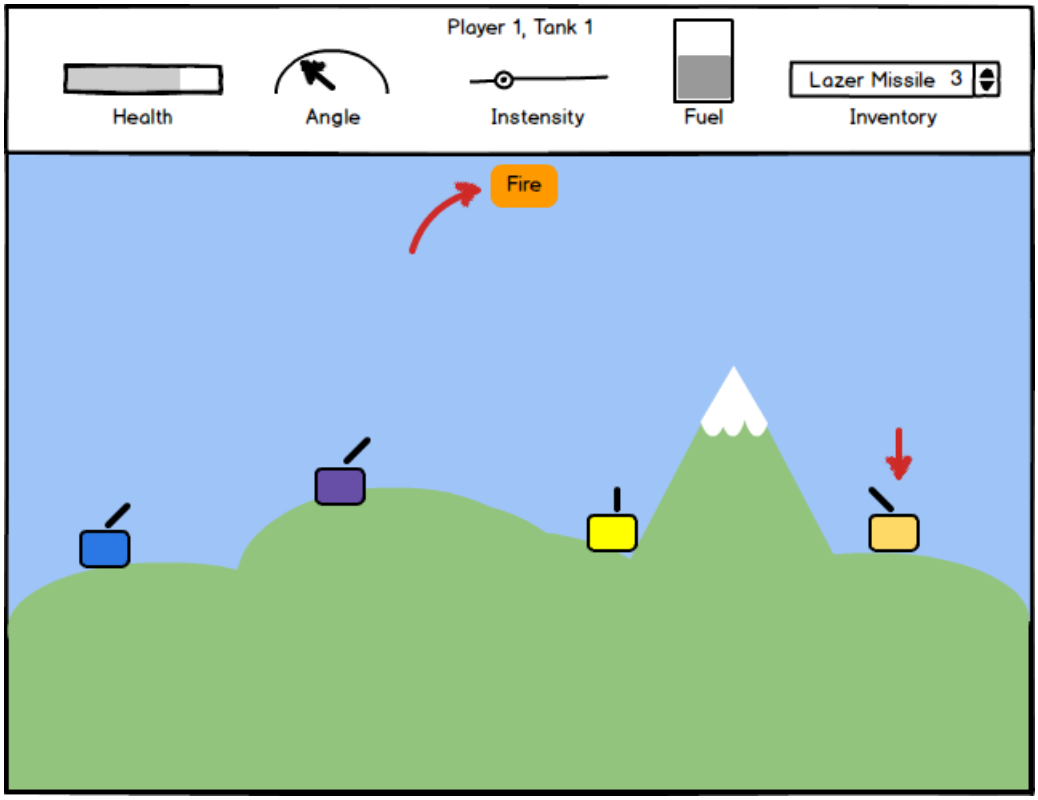
Project Description:

We're making a 2-player turn based projectile-oriented game similar to Tanks or Worms Armageddon. Each player controls one movable tank-like object on a randomly generated destructible terrain. Players can adjust angle, power, and ammunition used when firing upon the other player in an effort to bring the enemy's health to zero. This can be achieved through direct hits on the enemy or by destroying the ground underneath the enemy's tank.

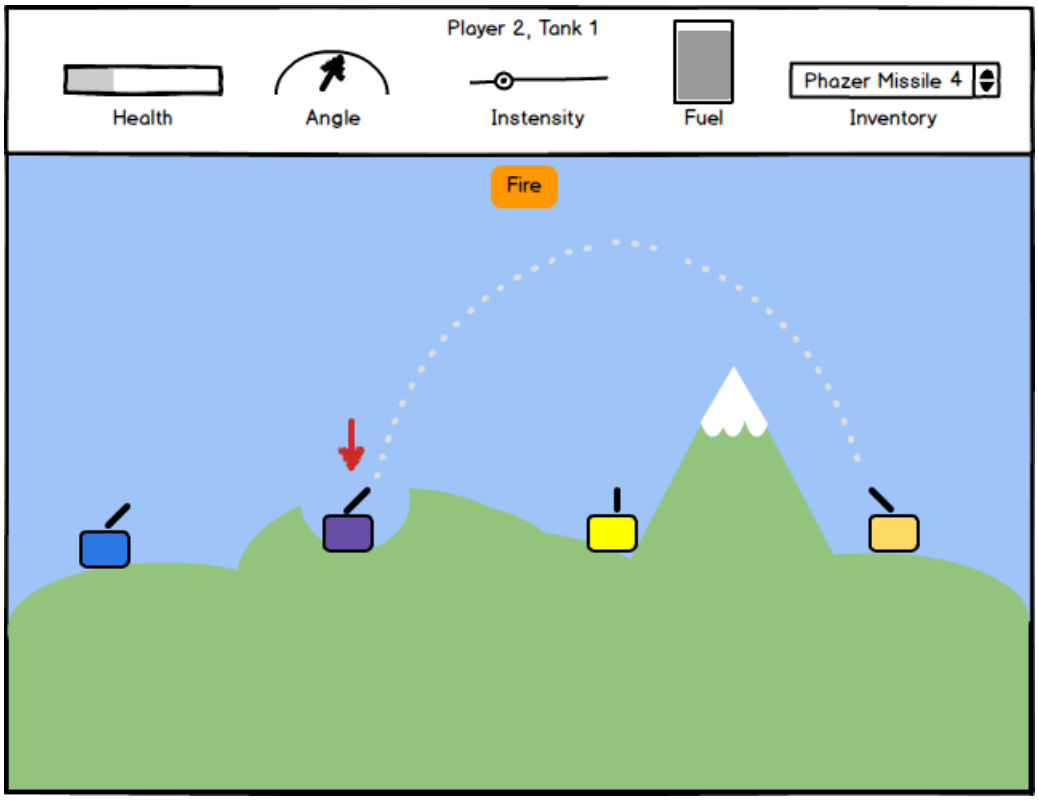
Features:

1. Randomly generates terrain within reasonable bounds.
2. Tanks move along surface of terrain.
3. Tanks can adjust firing angle and power.
4. Each player controls two tanks, both of which start on the same side of the board.
5. Players can purchase and use a variety of different weapons and defense mechanisms.
6. Users can view their ammunition inventory and select weapons to use.
7. Environment is destroyed when hit by projectiles and explosives.
8. Each player's health, firing angle and power, current weapon, fuel, and special items are displayed at all times.

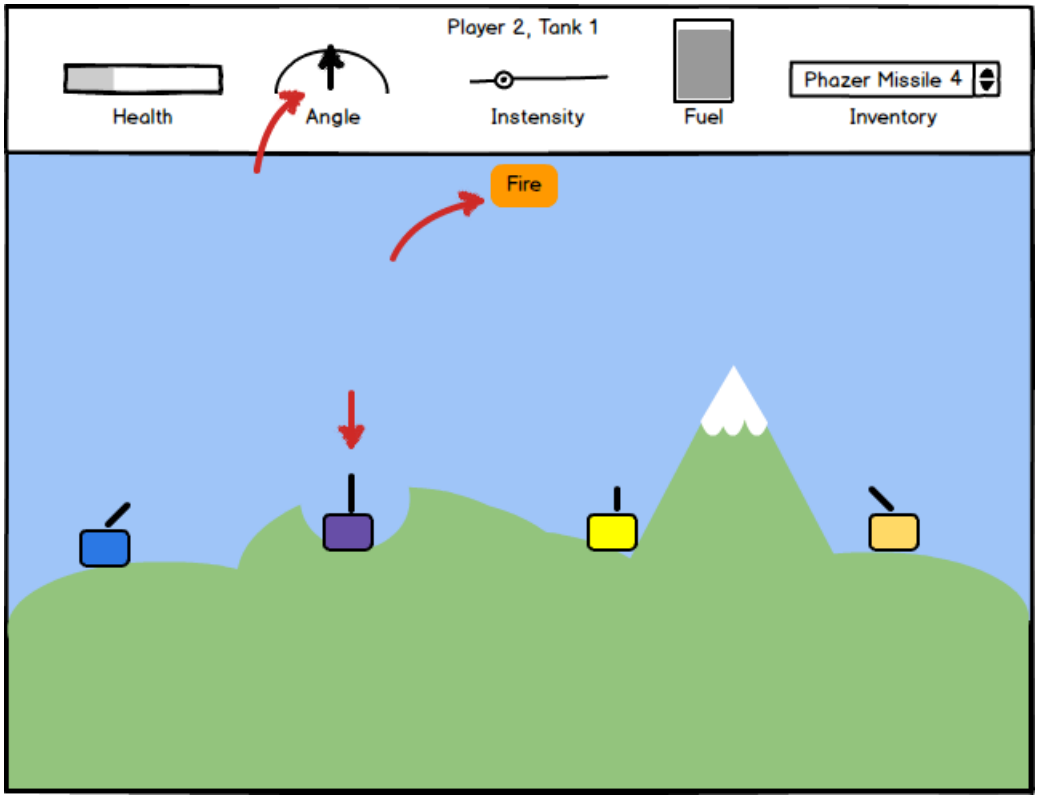
Storyboard:



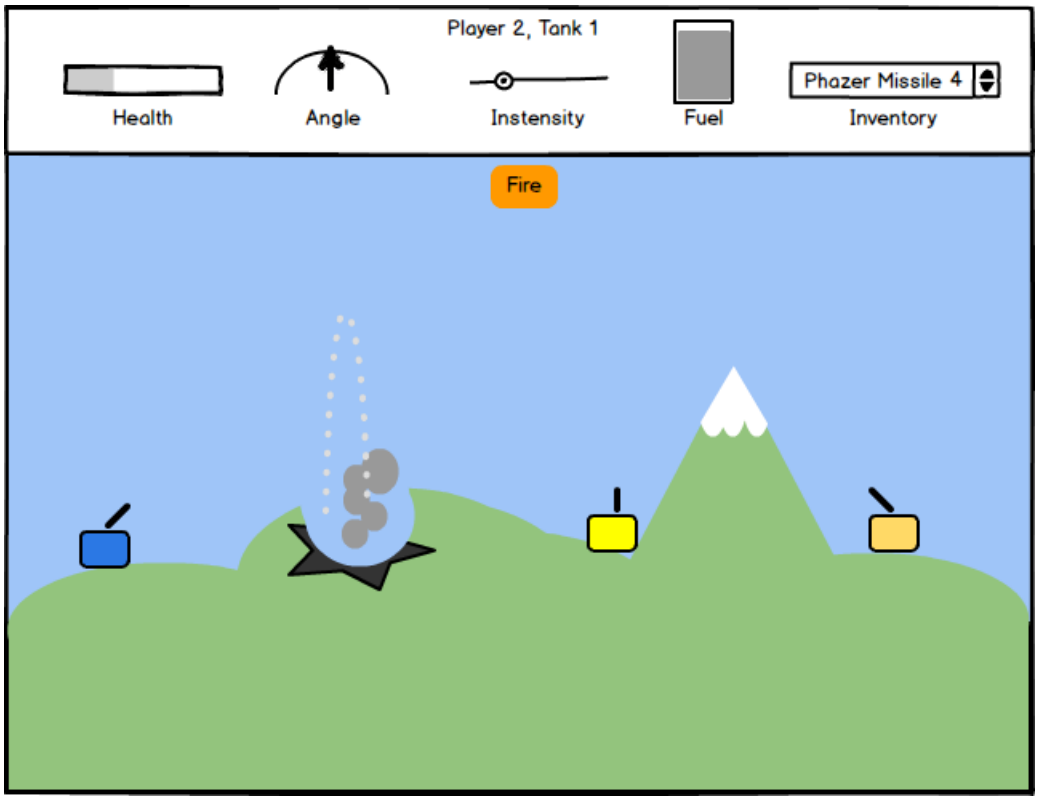
Player 1 and Player 2 are in the middle of a heated battle. Player 1 controls the orange and yellow tanks while Player 2 controls the blue and purple tanks. Player 1 fires a Lazer Missile at Player 2's tank.



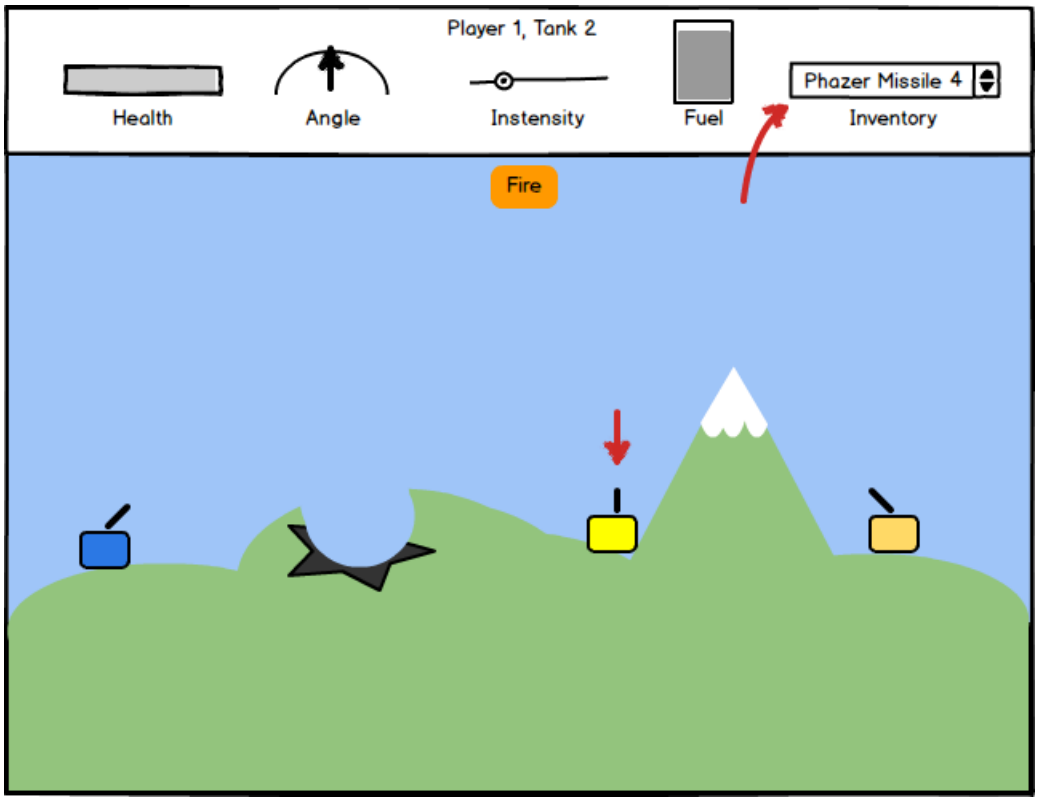
Player 1's tank has inflicted damage upon Player 2's tank, and has blown a crater in the earth below the tank. It is now Player 2's turn.



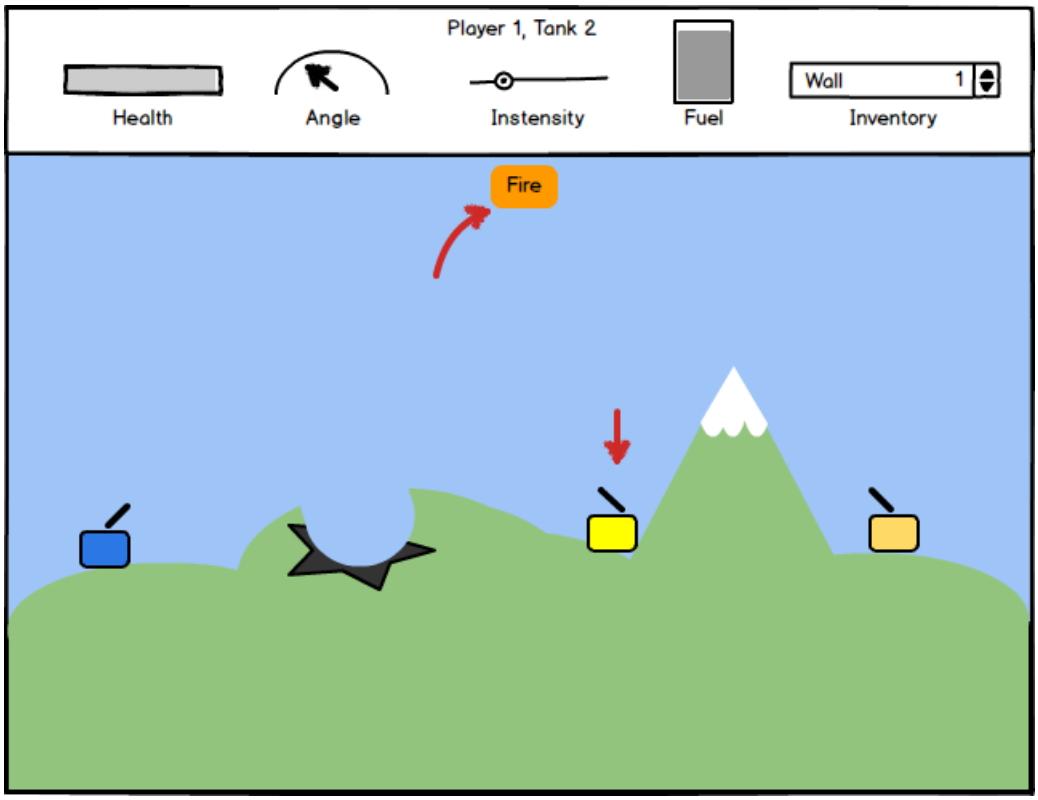
Player 2 adjusts the angle of his tank to straight up in the air! Let's see what happens...



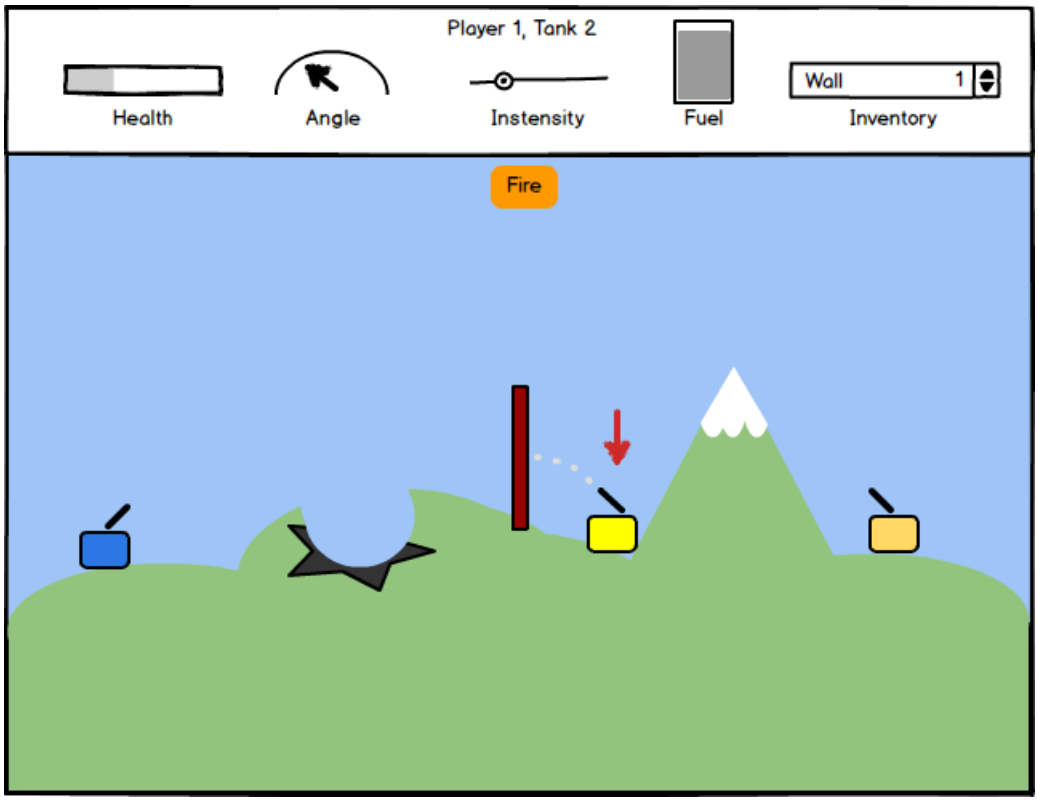
Player 2's missile went straight up in the air and destroyed his own tank. A poor strategic maneuver.



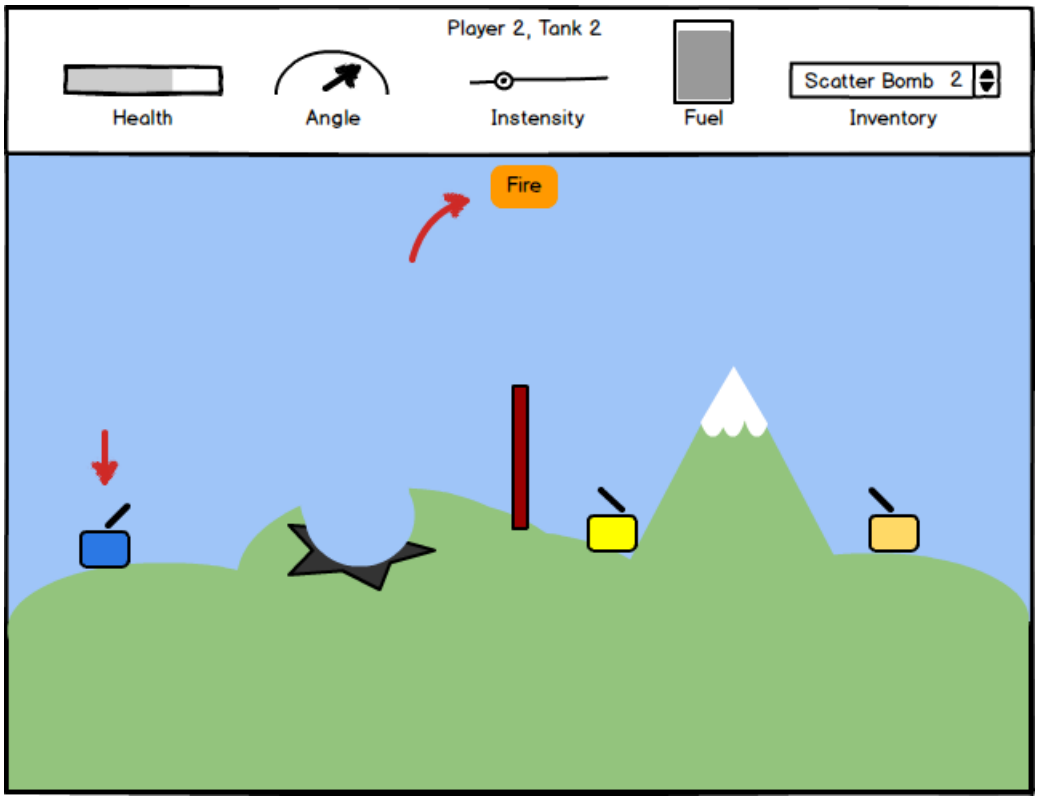
It is now back to Player 1's turn. She decides to search her inventory for the perfect tool...



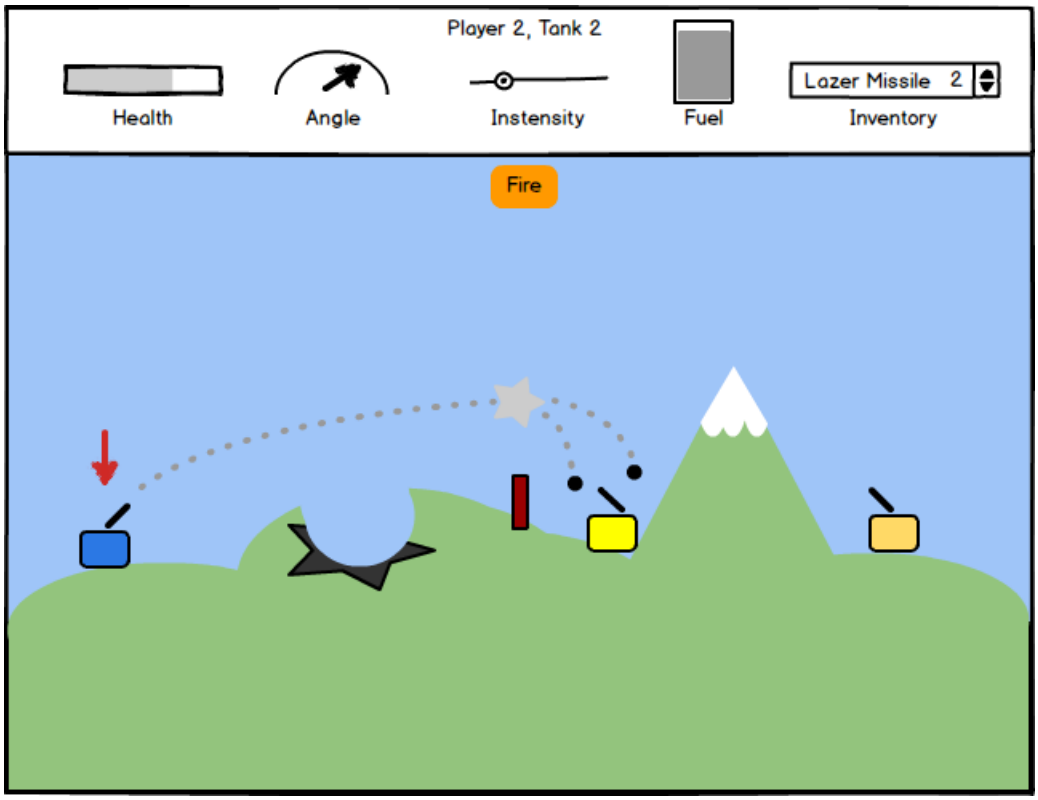
Player 1 wants to place a wall to block the attacks of Player 2's remaining tank, a defensive move.



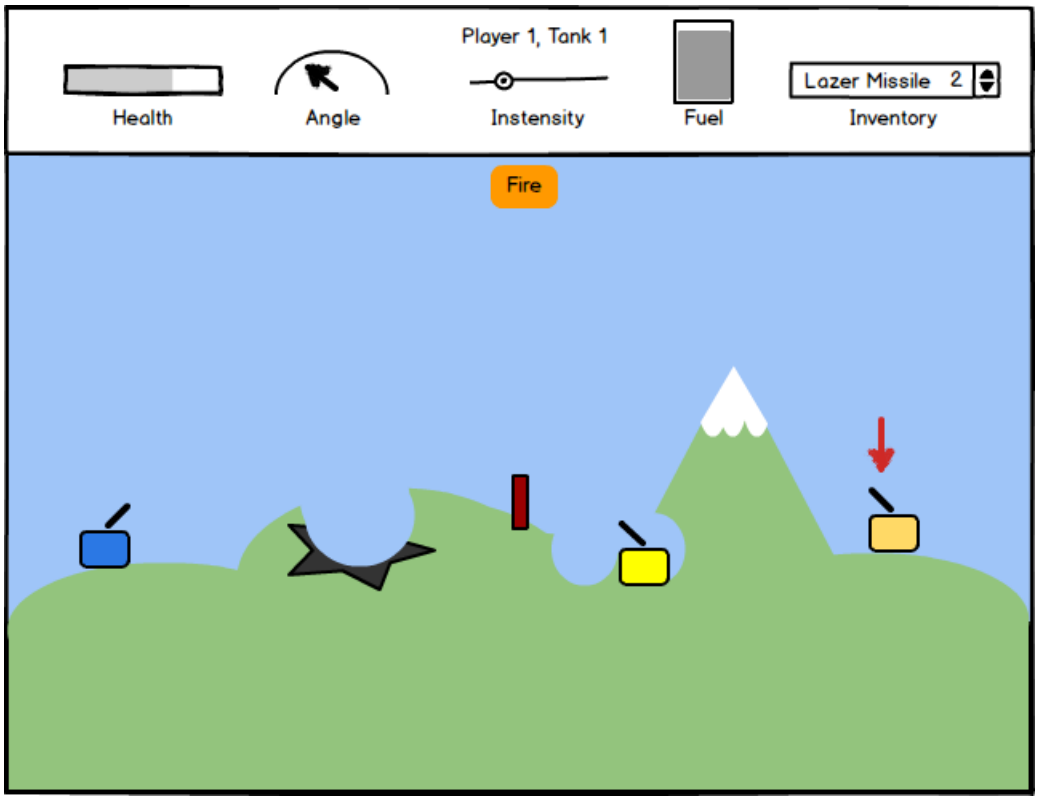
The wall is fired and lands, protecting Player 1's tank.



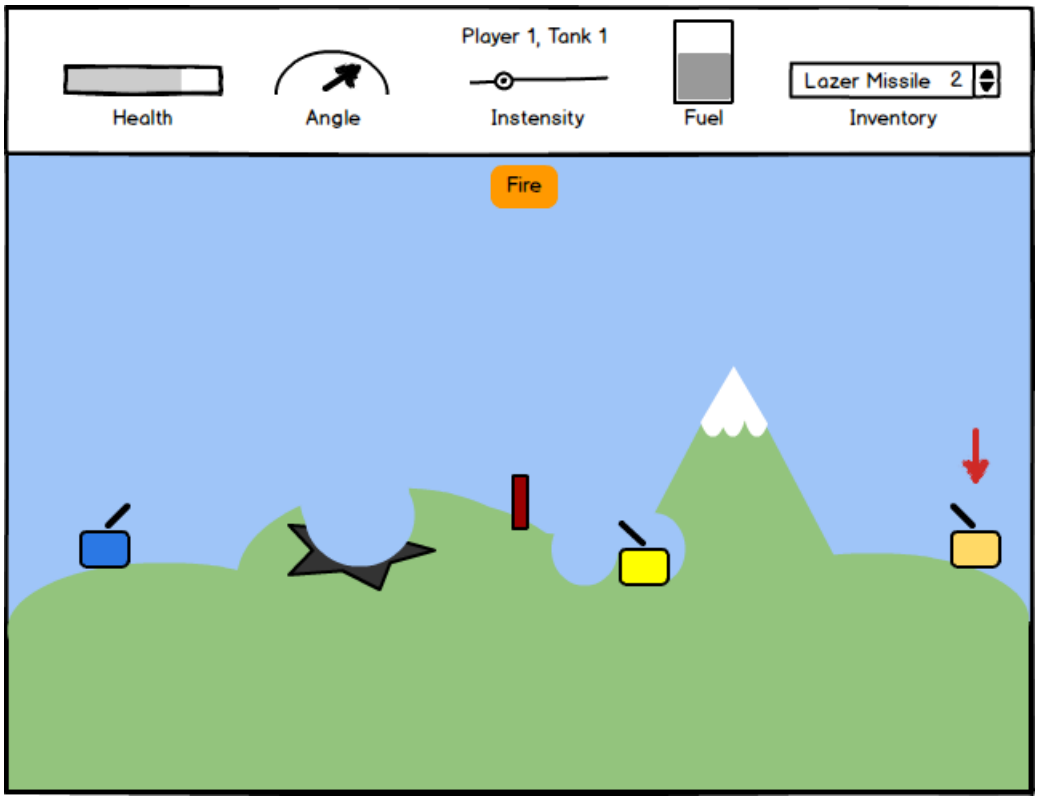
Player 2 chooses his Scatter Bomb weapon which creates more bombs upon impact and fires!



Player 2's Scatter Bomb destroys part of the wall. The remaining bombs rain a devastating barrage down upon Player 1's tank.

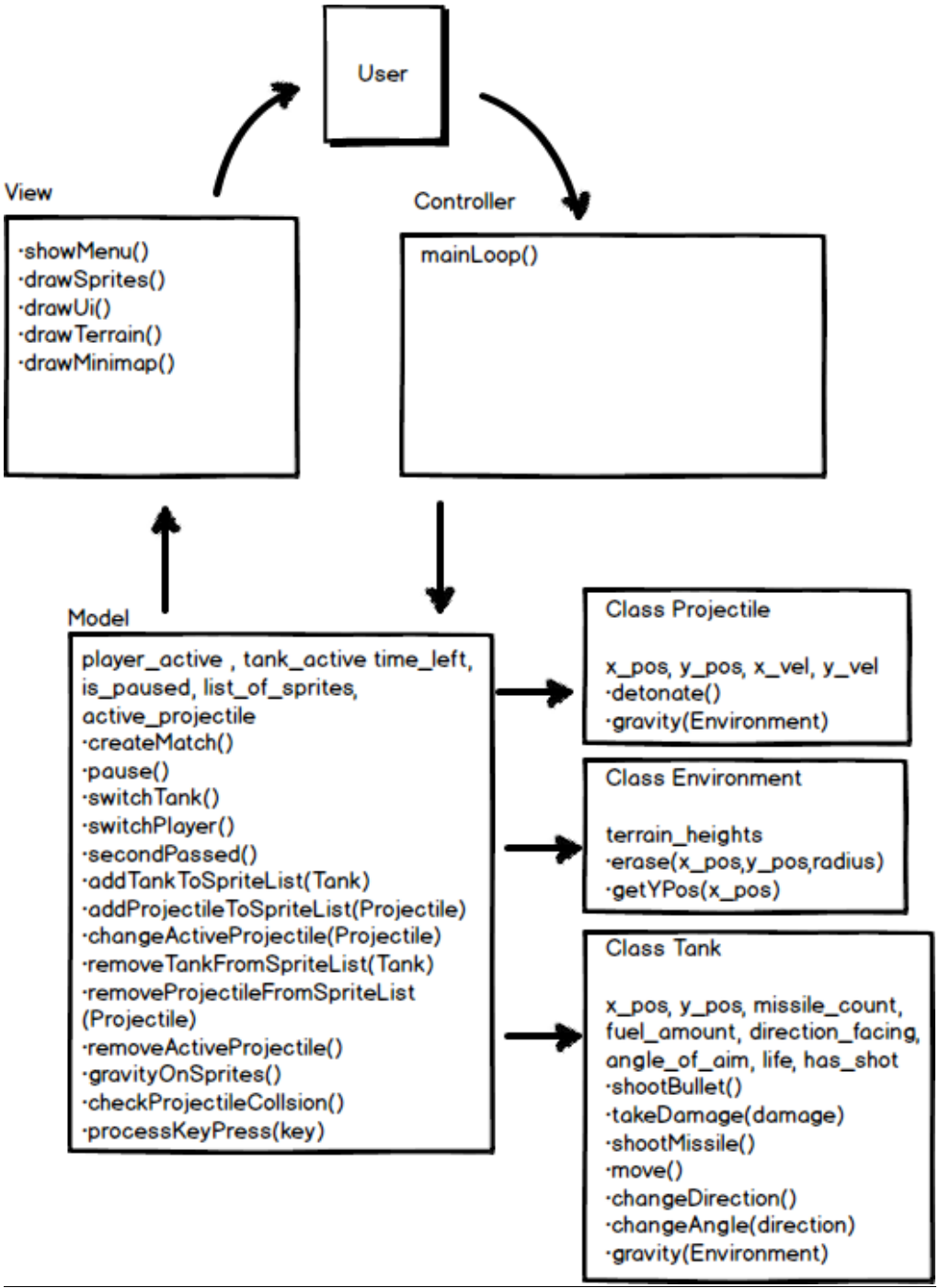


It's back to Player 1's turn. The ground below her second tank is obliterated, the tank is damaged.



Seeking a better angle to strike at Player 2 from, she uses precious fuel to move to a slightly better position. The game continues...

Software architecture:



Software architecture signatures

We will use a model-view-controller design to implement our game. Our controller will detect key presses to be passed to the model, which will subsequently affect the view. Our model will contain its own methods, as well as dependencies on the following classes:

class Tank: main class for player-controlled tanks. Tanks are able to create projectiles (bullets and missiles), and move around the map. They have a limited amount of fuel and can only move when active.
class Projectile: created by tanks (or randomly by the environment). Obey the basic Newtonian laws of physics, and can damage other tanks or explode pieces of terrain. Subclasses missile and bullet vary in strength.
class Environment: the battleground. Somewhat randomly generated, and can be destructed by projectiles. Holds up tanks.

Controller:

- mainLoop() – The main game loop that will run. Detects key presses to pass to model, and then makes calls to the view to update the view.

View:

- showMenu() – Shows the main menu when the game starts
- drawSprites() – Constantly checks on individual sprites and updates the view to reflect their positions
- drawUi() – Updates the user interface to display the user's current health, missile count, etc.
- drawTerrain() – Displays the environment at the beginning of the game, and updates the view when terrain is destroyed by a player
- drawMinimap() – Displays a small map that represents the positions of the various tanks on the larger map

Model:

contained within Model

- Instance variables:
  - player_active – indicates which player is currently "active" (it is his/her turn)
  - tank_active – indicates which tank is currently "active" (the player has selected it)
  - time_left – indicates how much time is left for the player's current turn
  - is_paused – indicates whether or not the game is paused

    - list_of_sprites – gives list of sprites that are active on the map (tanks and projectiles)
    - active_projectile – stores current projectile that is active on the map
- createMatch() – creates the terrain for a map and places player tanks on the map.
    - Unit tests: Call createMatch and see that a list of tanks are present in the SpriteList.
- pause() – pauses the game.
    - Unit test: call pause(), check the status of is_paused.
- switchTank() – changes tank_active when the player desires to change which tank he/she is in control of
    - Unit test: call switchTank(), check the status of tank_active to see if it switches to the correct tank
- switchPlayer() – changes player_active when the time for a given turn has run out
    - Unit test: call switchPlayer(), check the status of player_active to see if it switches to the correct player
- secondPassed() – changes time_left and determines if a player turn is over. Also makes calls to gravityOnSprites() and checkCollisions().
    - Unit test: Call secondPassed, see if time_left is affected.
- addTankToSpriteList(Tank) – adds a player tank to the Sprite list
    - Unit test: call method, and check to see that a tank instance is added to the spriteList array
- addProjectileToSpriteList(Projectile) – adds a projectile to the Sprite list
    - Unit test: call method, and check to see that a projectile instance is added to the spriteList array
- changeActiveProjectile(Projectile) – changes the active_projectile variable
    - Unit test: Call method, and check to see that the active_projectile variable is changed properly
- removeTankFromSpriteList(Tank) – removes a player tank to the Sprite list
    - Unit test: Call method, and check to see that the tank in question is removed from the spriteList
- removeProjectileFromSpriteList(Projectile) – removes a projectile to the Sprite list
    - Unit test: Call method, and check to see that the projectile in question is removed from the spriteList
- removeActiveProjectile() – changes the active_projectile variable to null
    - Unit test: Call method and check to see that whichever projectile is active is set to null
- gravityOnSprites() – makes calls to gravity() method on all sprites in the map.
- checkProjectileCollsion() – checks to see if projectile is in contact with a tank or the ground (in order to possibly cause detonations). Returns true/false.
    - Unit tests: A variety of tests with hard-coded instances of projectiles and tanks and see if it properly determines which ones have collided
        - Test1: tank at (250,40) and projectile at (250,80) – no collision

        - Test2: tank at (250,40) and projectile at (250,40) – collision
        - Test3: tank at (250,40) and projectile at (350,40) – no collision
        - Test4: projectile at (250,40) and terrain at (250,20) – no collision
        - Test5: projectile at (250,40) and terrain at (250,40) – collision
- processKeyPress(key) – determines which method to call (pause, move, shoot), dependent on key input.
    - Unit tests: variety of tests in different circumstances to see if the correct method is called.

class Projectile (subclass Bullet, Missile)

- Instance variables:
    - x_pos – x position of the projectile on the map,
    - y_pos – y position of the projectile on the map,
    - x_vel – current speed of the projectile in the x direction
    - y_vel – current speed of the projectile in the y direction
- detonate() – When a projectile is touching a tank or part of the terrain, it will explode, dealing damage to a tank that it is touching. Missile types will do additional damage in an area of effect and can explode part of the terrain.
- gravity(Environment) – Changes y_vel to account for gravity.
    - Unit tests: Variety of tests where the projectile should/shouldn't fall, check to see if y_vel is changed

class Environment

- instance variables:
    - terrain_heights – represented as an array, indicates the peaks of the terrain for a given environment. Changes in response to erase() and is used to inform the drawing of the map.
    - erase(x_pos,y_pos,radius) – erases a portion of the terrain, changing terrain_heights.
        - Unit tests: check terrain_heights has changed after a call to erase.
    - getYPos(x_pos) – returns the peak of the terrain for a given x_position to determine whether or not the tank/projectile is in contact with the terrain.
        - Unit tests: check to see that the correct peak within terrain_heights is returned.

class Tank

- **instance variables:**
  - x_pos – x position of the tank on the map,
  - y_pos – y position of the tank on the map,
  - missile_count – number of special missiles the tank has left,
  - fuel_amount – amount of fuel the tank currently has in its tank,
  - direction_facing – indication of whether the tank is facing left or right,
  - angle_of_aim – angle the tank is pointing its cannon,
  - life – amount of health the tank has left,
  - has_shot – keeps track of if the tank has fired during its turn

  Most of these methods will also have get and set methods associated with them.
- shootBullet() – creates and fires a bullet projectile based off the tank's current position and aim.
  - Unit test: Check to see that the instance of bullet is correct given the tank's current position and aim
- takeDamage(damage) – changes the amount of life for a tank, and possibly removes the tank if tank has 0 or less health.
  - Unit test: check to see that life is appropriately changed, and if necessary the tank is removed from the game.
- shootMissile() – creates and fires a missile projectile based off the tank's current position and aim.
  - Unit test: Check to see that the instance of missile is correct given the tank's current position and aim
- move() – moves the tank in a certain direction (costing fuel) based off the tank's current position and direction, affecting x_pos and y_pos if applicable.
  - Unit test: Check to see that when move() is called, x_pos, y_pos and fuel_amount are updated correctly (if move is legal)
- changeDirection() – changes the attribute direction_facing.
  - Unit test: Check to see that when changeDirection() is called, the direction is changed
- changeAngle(direction) – changes the attribute angle_of_aim.
  - Unit test: Check to see that when changeAngle() is called, the angle_of_aim is changed.
- gravity(Environment) – Changes y_vel to account for gravity.
  - Unit tests: Variety of tests where the tank should/shouldn't fall, check to see if y_vel is changed