Vertex TD

Michael (Mengyuan) Liu  
[Michael.Liu0@gmail.com](mailto:Michael.Liu0@gmail.com)

Individual Project

Web Multimedia  
INFO 3225

Abstract

There is an old Tower Defense game in the days of online flash game portals called Vector TD. It had a clean, minimal aesthetic in contrast to a lot of very flashy Tower Defense games available at the time.

I decided to clone the game for this project mainly because it fits very well into the required scope and functions – inherited object hierarchies, Matrix transformations, and UI requirements such as mouse and keyboard interactions.

Making Of

The program consists of two main parts, the Level Editor, and the game mechanics. For simplicity reasons I tackled the Level Editor first.

The level editor consisted mostly of laying the foundation of displaying the level grid, as well as displaying the right-side banner with a proper grid layout of buttons. Simple logic for finding the correct grid under the mouse was implemented. The final two things implemented were unique to the Level Editor: Drawing enemy paths and saving the level. For time constraint and UI reasons I cut a feature to customize enemy wave settings.

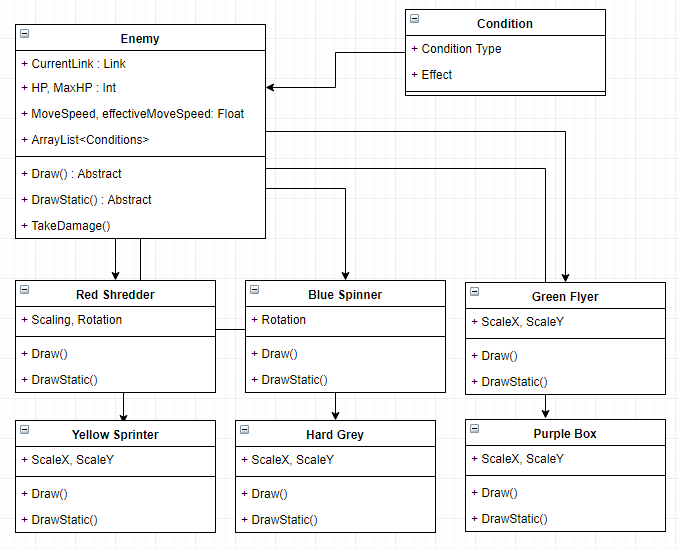
The game mechanics are much more complicated. Selecting towers to be built was a simple button implementation, but coding everything in a generic way proved to be impossible without much more time simply due to not being able to access PApplet properly within Processing unless we run it as a full Java program. I opted to instead write everything as an Object instead of trying to use Static functions.

All said towers wasn’t terribly difficult to implement, nor were enemy type behaviours; what did prove to be time consuming was tower shooting behaviours. Since it had to be displayed with an eye to UX, I couldn’t simply draw lines for one frame at a time, and lines had to be faded in and out which proved somewhat cumbersome. Having never really coded at such a low level (since I usually use Unity’s built in functions) it impresses upon me how much coding is running under the hood at any given time.

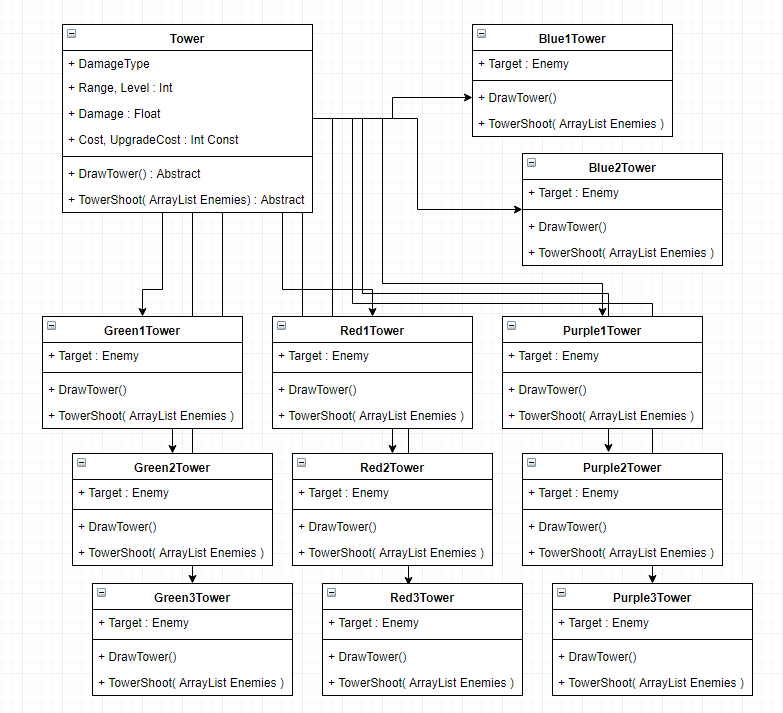
The very last thing to implement was contextual menus that only show up once a tower is selected, which lead to selectively rendering buttons. Pretty sure I used too many IF conditionals but at this point I was pressed for time and did not want to expend extra energy to code in a clean way. I also discovered I forgot to use Utility classes such as PVectors to my advantage, and instead having to re-implement useful functions such as calculating distance.

Tuning took the rest of the time. Player states such as Money and win/loss states was implemented. The game was playable.

Enemies



Enemies are simple, they share the vast majority of their fields. Movement are taken care of generically in the Enemy class, so the individual enemy is only responsible for drawing their representations (Appendix D). Movement simply consists of moving along an array of tiles, until it reaches the end.

Towers

There are 4 types of towers, each have 3 types (except blue), that’s 11 towers. Luckily most isn’t very different others of the same type, allowing for code reuse.

Drawing towers is easy, usually consisting of a few lines. Drawing the shooting animation is vastly more difficult (an example can be found in Appendix B). Finding enemy targets is achieved using a brute force method of iterating over all enemy objects, and depending on the targeting type different enemies are picked (however re-targeting is not engaged until the enemy moves out of range or is dead) (Appendix B). Extra fields for line fading (Appendix C) and other PVectors for consistent displays took further development time.

The result is a generic system for drawing towers and shooting mechanics without having to worry about implementation details, unlike the implementation of buttons which used a ENUM and SWITCH in the main loop.

Conclusion

This was a valuable learning experience. 30 hours of work should have produced more work than this, and is perhaps attributed to my inexperience in using Processing as well as not using external libraries for UI elements (such as G4P or ControlP5) necessitating a clumsy UI implementation process. Four towers had to be cut due to time constraints, and I don’t believe the game is particularly balanced well, even if it plays okay at first glance. Overall further work is needed.

Appendix A

if( target == null ){

for( Enemy e : enemy ){

if( FindDistance( location, new PVector( e.x, e.y ) ) > getRange() ){

continue;

}

if( target == null ){

target = e;

continue;

}

switch( targetingType ){

case Hard:

if( e.hp > target.hp ){

target = e;

}

break;

case Weak:

if( e.hp < target.hp ){

target = e;

}

break;

case Close:

if( FindDistance( location, new PVector( e.x, e.y ) ) < FindDistance( location, new PVector( target.x, target.y ) ) ){

target = e;

}

}

}

if( target == null ){

return;

}

}

Appendix B

public void DrawTowerShoot( ArrayList<Enemy> enemies ){

for( HashMap.Entry<Missile2, Enemy> entry : missiles.entrySet() ){

if( MoveTowards( entry.getKey(), entry.getValue() ) ){

toRemove.add( entry.getKey() );

}

}

for( Missile2 mis : toRemove ){

missiles.remove( mis );

}

toRemove.clear();

if( reload > 0 ){

reload--;

} else {

PVector location = TranslateGridToPixel( tile.x, tile.y );

inRange.clear();

for( Enemy e : enemies ){

if( location.dist( new PVector( e.x, e.y ) ) < getRange() ){

inRange.add( e );

}

}

if( inRange.isEmpty() ){

return;

}

missiles.put( new Missile2( new PVector( tile.x \* LEVEL\_TILE\_SIZE + OFFSET\_X + fireSpot[currentSpot].x, tile.y \* LEVEL\_TILE\_SIZE + OFFSET\_Y + fireSpot[currentSpot++].y ) ), inRange.get( int( random( inRange.size() - 1 ) ) ) );

if( currentSpot > 8 ){

currentSpot -= 9;

}

reload = 2;

}

}

private boolean MoveTowards( Missile2 missile, Enemy target ){

if( target.hp <= 0 ){

return true;

}

PVector move = new PVector( target.x + LEVEL\_TILE\_SIZE/2, target.y + LEVEL\_TILE\_SIZE/2 ).sub( missile.location );

if( move.mag() < 8 ){

target.TakeDamage( DamageType.Red, getDamage(), null );

return true;

}

move.setMag( 8 );

missile.location.add( move );

noStroke();

ellipseMode( CENTER );

fill( GAME\_TOWER\_RED\_COLOR );

ellipse( missile.location.x, missile.location.y, 4, 4 );

return false;

}

Appendix C

private void DrawBeam( PVector location, PVector target ){

pushMatrix();

translate( HALF\_TILE\_SIZE, HALF\_TILE\_SIZE );

stroke( GAME\_TOWER\_RED\_COLOR, 180 - (( 8 - transTickdown ) \* 40 ) );

strokeWeight( 3 );

line( location.x, location.y, target.x, target.y );

strokeWeight( 1 );

popMatrix();

}

Appendix D

pushMatrix();

translate( x, y );

// Draw sprite

// Draw circle part

noFill();

stroke( #F20202 );

ellipseMode( CENTER );

ellipse( LEVEL\_TILE\_SIZE/2, LEVEL\_TILE\_SIZE/2, 6, 6 );

pushMatrix();

int halfTile = LEVEL\_TILE\_SIZE/2;

translate( halfTile, halfTile );

scale( constrain( randomGaussian() + 0.7, 0.8, 1.5 ) );

rotation += random( 0, 40 );

if( rotation > 360 ){

rotation -= 360;

}

rotate( radians( rotation ) );

// Draw spikes

line( -3, 0, -10, 0);

line( 3, 0, 10, 0 );

line( 0, -3, 0, -10 );

line( 0, 3, 0, 10 );

rotate( PI / 4 );

line( -3, 0, -6, 0);

line( 3, 0, 6, 0 );

line( 0, -3, 0, -6 );

line( 0, 3, 0, 6 );

popMatrix();

popMatrix();