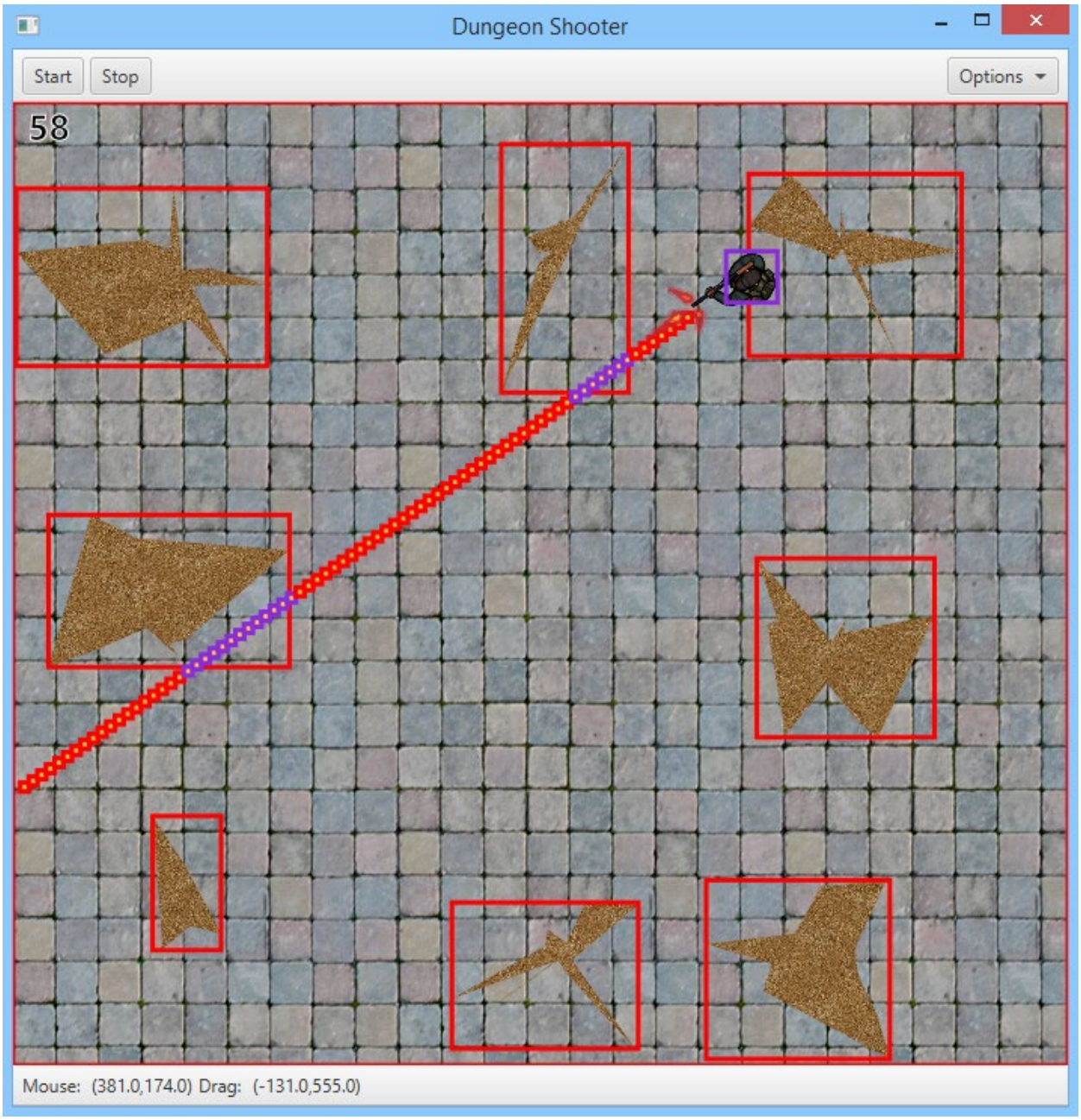
Assignment



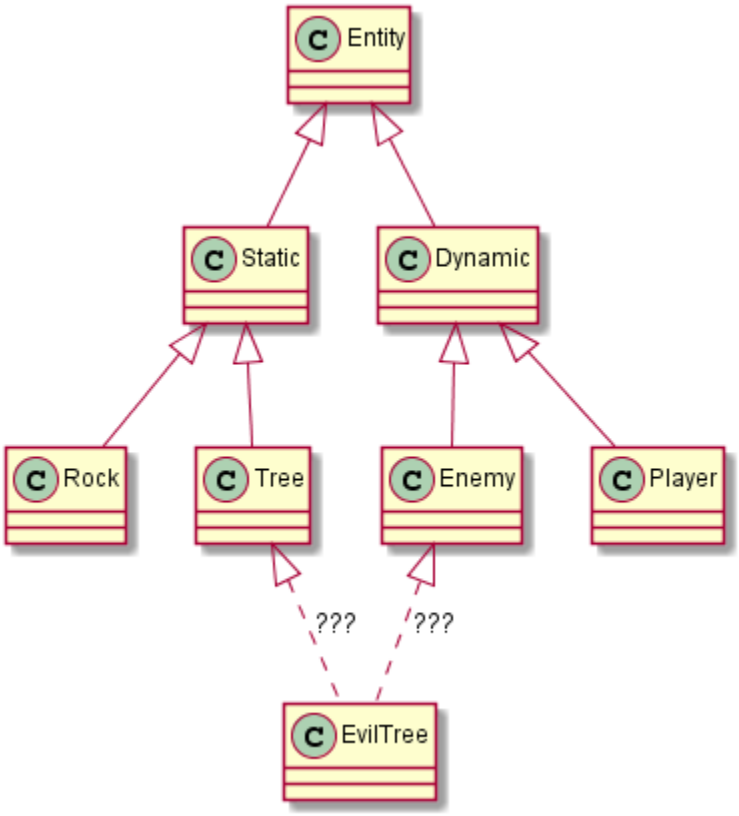
I recommend reading the two links bellow if you like better understanding of simple game design. Read the first on for sure if you are going to read only one.

1. https://www.gamedev.net/articles/programming/general-and-gameplay-programming/understanding-component-entity-systems-r3013/
2. https://www.gamedev.net/articles/programming/general-and-gameplay-programming/the-entity-component-system-c-game-design-pattern-part-1-r4803/

Assignment

Design

In this assignment we going to focus more on composition over inheritance. Look at the diagram below taken from gamedev.net [1].

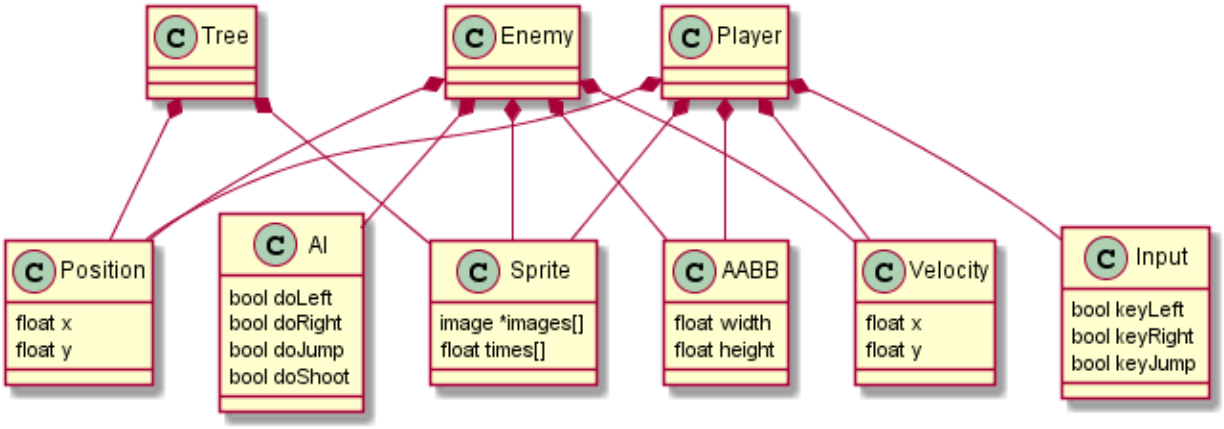


what is wrong with this system? It looks easy enough however the more complex your system becomes the harder it will be to expand your code. In this example what is EvilTree? For solving this you can inherit form one of Enemy or Tree but then you must duplicate much of your code to get the behavior you need. And this problem will continue to grow.

Now instead of inheritance if we go with composition, we can solve this issue and create a more expandable system. This DP is called Entity component system which is an architectural design pattern.

Entity component system

In this design pattern instead of completely relying on inheritance we rely more on composition. As you can see in diagram bellow entities are composed of traits that can define them. in this assignment our traits are HitBox, Sprite, InputAdapter and to some level Point.



Assignment

Class and Sequence Diagrams

Before starting the assignment make sure to spends some time and properly review all class diagrams. It is important to be aware of all relationships between classes and methods available in each class.

Many components in diagrams are not mentioned in the details bellow, you should be able to infer those details.

DungeonShooter

1. This class is the same as RayCast.
2. Toolbar at the bottom is identical to RayCast.
3. ToolBar at the top is like RayCast but with less components.
    a. The only options in the dropdown menu are FPS and Bounds
    b. There will be only one animator so no need for the AnimatorBox. However, if desired customize your old lab so you can just add the new animator to it.
4. Look at the Sequence Diagram to create your init() method.
5. When making the player use 70 and 46 as width and height.

CanvasMap

1. Very similar to lab.
2. Cleanout the extra properties as only fps and bounds are needed.
3. Initialize the 4 lists. Initlize buffer and projectiles to size 500, static shapes to 50 and player to 1.
    a. Players list will hold the player/s and other npc/s.
    b. Staticshapes list holds only polyshapes. Which are obstacles on the map.
    c. Projectiles list holds bullets that have been fired and any other projectile.
    d. Buffer list holds entities that need to be added to projectile list. Have in mind you cannot modify lists actively while trying to modify it such as remove operation. Buffer list is used to keep track of the entities to be added to projectiles list at the beginning of each frame. This can be done by calling updateProjectilesList method which simply adds the buffer to projectiles and then clear the buffer.
    e. Border is of type Polyshape which is your background and general shape of your map. It can be simply a square you can make it more complicated.
    f. By calling:
    border.getDrawable().setFill(new ImagePattern(new
    Image("file:assets/floor/pavin.png"), 0, 0, 256, 256, false));
    you can use the assets given to you for background.
4. Unlike lab the Canvas is set through a setter method. It is created outside and passed to CanvasMap through setter.
5. setDrawingCanvas sets the new canvas.
    a. check if canvas is null, if it is throw null pointer exception.
    b. We want the size of the border to change if the size of canvas changes. Get the width and height property of canvas and add to each a listener to update the points in border class. You can just call set points and set the four corners again. if you decide to go for a shape which is not a simple rectangle you need to customize this behavior for your need. You can try to add a scrolling page map.

6. fireBullet is called by player when left mouse is clicked. Pass the bullet argument to buffer. In a more complicated game this will be the job of a projectile manager. However, for the sake of simplicity we are mixing that job with canvasmap. If you decide to add any other type of projectiles you still can use this same method.
7. updateProjectilesList is called to add the buffer to projectiles list and then clear the buffer.
8. inMap is used to call check if the HitBox argument is still within the boarder.getHitBox().

AbstractAnimator

1. this class does not contain the intersect calculation anymore That method is moved to utility in a class called IntersectUtil.
2. Mouse movement methods have also been removed and placed in InputAdapter.
3. Handle method has been simplified as now it only needs to deal with fps. Other options are removed.
4. A new method drawEntites is added. Details are in sequence diagram.
    - when drawing order matters. If you draw canvas last it will cover all other entities.

Animator

1. details are in sequence diagram. The job of this animator is to update all entities first and then draw them. during the update all entities will be looped. Each entity has a update method that needs to be called. After the update all interactions between entities must be checked.
    a. Are the entities still in map? if not remove them.
    b. Have the projectiles hit anything such as static shapes? If so remove them.
    c. Has the player hit a shape? If so, do not go through the shape and step back.
    d. If there are other entities that can attack to be attacked must be checked.
    e. If bounds option is active change color of bounding box to something else if near shape.

Bullet

1. This class is an entity. Meaning it must implement Entity
2. Load the static final BULLET image using:
   new Image("file:assets\\bullet\\b_3.png");
3. There are 2 constructors. One constructor does not take width and height. When chaining to next constructor use 6 and 6 as width and height.
    a. Angle is defined by the position of mouse relevant to center of player which is given by player.
    b. Create a new hitbox and setBounds with given x,y,w,and h.
    c. Create a new sprite like this:

```
sprite = new Sprite(){
   private RectangleBounds bounds = hitbox.getBounds();
   public void draw( GraphicsContext gc){
      gc.drawImage( BULLET, bounds.x(), bounds.y(), bounds.w(), bounds.h());
   }
};
```

4. In the update method you need to calculate the x and y like:
   double x = Math.cos(Math.toRadians(angle)) * 7;
   double y = Math.sin(Math.toRadians(angle)) * 7;
   hitbox.translate(x, y);
   7 is the speed at which the bullets travel. Feel free to customize it. Finally translate the hitbox to the new position. Have in mind translate adds x and y to current position, it is not a set.

Entity

1. This is an interface used by all entities in the game. Look at class diagram for details

FpsCounter

1. Same as the lab

Player

1. Player class represents one player, the code however can be useful for npc/s as well.
2. rotationPlayer class variable is from javafx used for matrix rotation in 2d. if you want to know more read the java doc for javafx.scene.transform.Rotate.
3. player has a reference to CanvasMap just for spawning bullets.
4. Angle in degree is to determine the relation of player muzzle to mouse.
5. Player and muzzle frame are just for animation as it rotates between images to create an animation. Images are in assets folder.
6. Prev is to old the last valid position of player.
7. In constructor initialize values
   a. Create a rotation player with default constructor.
   b. Create a pos with point. We going to treat player as a square. Ignoring the arm. So the center/pos is x-w/2 and y-5/2
   c. Use the point copy constructor to create a new prev
   d. Create dimension with w and h
   e. Create a sprite using code in next page.
   f. Create a hitbox using code below:
      double size = h * .74;
      hitbox = new HitBox().setBounds(pos.x() + dimension.x() * .303 - size / 2, pos.y() + dimension.y() * .58 - size / 2, size, size);
      this calculation find the relative center location of the specific image assets provided with the assignment. If you want to customize your assets you have to change this calculation.
8. These methods calculate centers and muzzle. The hardcoded numbers are position ratios in the image asset.
   i. getPlayerCenterX(){ pos.x() + dimension.x() * .303; }
   ii. getPlayerCenterY(){ return pos.y() + dimension.y() * .58; }
   iii. getRifleMuzzleX(){ return pos.x() + dimension.x() * .93; }
   iv. getRifleMuzzleY(){ return pos.y() + dimension.y() * .73; }

9. stepBack function is used when player hits a wall. In the method call undoTranslate on hitbox and then call move on pos and pass prev to it. This will undo last move.
10. calculateAngles is used to find which direction the bullet should fly if any and what directly the player image should be rotated to. Angle can be found using: Math.toDegrees(Math.atan2(input.y() - getPlayerCenterY(), input.x() - getPlayerCenterX())); then call setAngle, setPivotX and setPivotY on rotationPlayer and pass angle, getPlayerCenterX() and getPlayerCenterY();

```
sprite = new Sprite(){
   //player and muzzle each have 20 and 16 set of images than can be loaded
   private final Image[] PLAYER = new Image[20];
   private final Image[] MUZZLE = new Image[16];
   {
      //load the images
      for( int i = 0; i < PLAYER.length; i++){
         PLAYER[i] = new Image( "file:assets\\rifle\\idle\\survivor-idle_rifle_" + i + ".png");
      }
      for( int i = 0; i < MUZZLE.length; i++){
         MUZZLE[i] = new Image( "file:assets\\muzzle_flashs\\m_" + i + ".png");
      }
   }
   public void draw( GraphicsContext gc){
      gc.save();
      //rotate gc for drawing
      gc.setTransform( rotationPlayer.getMxx(), rotationPlayer.getMyx(),
         rotationPlayer.getMxy(), rotationPlayer.getMyy(),rotationPlayer.getTx(), rotationPlayer.getTy());
      //if left click display fire animation
      if( input.leftClicked()){
         gc.drawImage( MUZZLE[(int) muzzleFrame], getRifleMuzzleX() - 8, getRifleMuzzleY() - 25, 50, 50);
         //this number is how fast the next frame of fire animation will be drawn. The higher the faster.
         muzzleFrame += .5;
      }
      //darw player image
      gc.drawImage( PLAYER[(int) playerFrame], pos.x(), pos.y(), dimension.x(), dimension.y());
      gc.restore();
      // this number is how fast the next frame of player animation will be drawn. The higher the faster.
      playerFrame += 0.25;
      //reset frame counts if reach the max frame
      if( playerFrame >= PLAYER.length){
         playerFrame = 0;
      }
      if( muzzleFrame >= MUZZLE.length || !input.leftClicked()){
         muzzleFrame = 0;
      }
   }
};
```

11. in update use the code bellow to fire the bullet:
    Point2D muzzle = rotationPlayer.transform(getRifleMuzzleX(), getRifleMuzzleY());
    map.fireBullet(new Bullet(this.angle, muzzle.getX(), muzzle.getY()));

PlayerInput

1. this class uses InputAdapter. Adapter class has majority of javafx code in it. An instance of InputAdopter is passed through the constructor. In the constructor call forceFocusWhenMouseEnters, registerMouseMovment, registerMouseClick and registerKey on adapter and pass appropriate methods from your PlayerInput class to it using method reference ::, like this::moved and this::dragged.
2. moved and dragged methods are overridden to record x and y of mouse.
3. keyPressed and keyReleased are overridden to call changeKeyStatus. Pass to changeKeyStatus key.getCode() and true of pressed else false.
4. In changeKeyStatus use a switch to check if W (up), A (left), S (down), D (right), Shift or Space is press if so, update the Boolean value for that class variable with Boolean received from the argument.
5. In mouseReleased set all mouse related Booleans to false and then call super.mouseReleased(e). super method is not empty so it should be called to complete the code.
6. In mousePressed update x and y using the get methods in MouseEvent. Then update the mouse related Booleans using e.isPrimaryButtonDown(), e.isSecondaryButtonDown() and e.isMiddleButtonDown(). Finally call super method is not empty so it should be called to complete the code.
7. In leftOrRight and upOrDown if neither Boolean is true return zero meaning nothing should change in x or y direction. If right or down are true return 1 else -1.
8. hasMoved is a quick checker which returns true of any of the direction Booleans are true.

PolyShape

1. this class implements entity, you can use most of the code from your lab. Remove Colors, bound and strokeWidth class variables. Instead add Hitbox and Sprite.
2. In constructor initialize hitbox and sprite using the code in next page.
    a. Hit box can be customized for custom shapes like PolyShape. By overriding hasIntersectFull method and returning true hitbox will check for more detailed points which is returned in getPoints which also must be overridden.

Sprite

1. This class implements Drawable< Sprite>. This class is abstract and does not provide a definition for draw method forcing any class that uses it to implement draw.
2. This class will also hold class variables of type Paint called fill and stroke plus strokeWidth of type double.
    a. fill and stroke values in this class are of type Paint. this allows the user to set any Color to the sprite or using the ImagePattern class to choose an image asset and set it as filling. Ex:
    setFill(new ImagePattern(new Image("file:assets/concrete/dsc_1621.png")));

Assignment

Drawable

1. This interface is used by sprite as guidline. used to be polyshape and fps counter.

```
// Code in Constructor of PolyShape
hitbox = new HitBox(){
   protected boolean hasIntersectFull(){
      return true;
   }
   protected double[][] getPoints(){
      return points;
   }
};
sprite = new Sprite(){
   {
      setFill( new ImagePattern( new Image( "file:assets/concrete/dsc_1621.png")));
   }
   public void draw( GraphicsContext gc){
      gc.setLineWidth( getWidth());
         if( getStroke() != null){
            gc.setStroke( getStroke());
            gc.strokePolygon( points[0], points[1], pointCount);
         }
         if( getFill() != null){
            gc.setFill( getFill());
            gc.fillPolygon( points[0], points[1], pointCount);
         }
      }
};
```

HitBox

1. this class implements Entity.
2. Prev is the last valid position of hitbox.
3. Points[][] is used to keep reference of points for fine tone checking of intersects. This exits to prevent creating a new array every time. Have in mind before using this class variable you MUST call getPoints() everytime to update the points to the newest one.
4. Result[] exits for the same reason as points.
5. In constructor initialize points to size 2 by 4 and results to 4. Create prev point as well.
    a. Create sprite:

```
sprite = new Sprite(){
   public void draw( GraphicsContext gc){
      gc.setStroke( getStroke());
      gc.setLineWidth( getWidth());
      gc.strokeRect( bounds.x(), bounds.y(), bounds.w(), bounds.h());
   }
};
sprite.setStroke( Color.RED).setWidth( 3);
```

6. setBounds set the bound if the argument is RectangleBound otherwise create the rectangle using the x, y, w, and h then call setBounds
7. every time translate is called you call prev.move(bounds.startPos()); first then call translate on bounds and pass dx and dy.
8. undoTranslate calls move on bounds and passes prev to it. This sets the position to previous state.
9. containsBounds checks if a hitbox is completely within current hitbox. This can be done bay calling contains method of bounds and passing to it hitbox.getBounds();
10. intersectBounds check if a hitbox is within and or overlapping the current box. Same as last method but instead use intersects of bounds.
11. intersectFull should be called after intersectBounds has returned true as this method is more expensive. With in this method call intersectFull(box.getPoints());
    a. in protected intersectFull check for intersection between all line segments of points in current hitbox and points passes as argument.
        i. This method will have 2 for loops one nested in other. Outer loop will go through current hitbox points while inner loop goes through otherpoints.
        ii. In the inner loop IntersectUtil.getIntersection() method to check if lines cross.
        iii. If the result of IntersectUtil.getIntersection() is true check to see if result[2] <= 1 is also true. We want to make sure lines are crossing each other on segments specified. If both are true return true otherwise continue looking.
        iv. Return false if nothing found.
12. hasIntersectFull returns false by default as unless getPoints is overridden like in PolyShape the methods in RectangleBound is enough for checking intersects.
13. getPoints can return bounds.toArray(points); by default so if it is called it won't be null;
14. update is empty.

Bonus

This is a big assignment that has room for grow in many aspects. Here are some bonus suggestions, difficulty increasing in ascending order:

1. add a score system to game, fake at first unless you added more bonusses.
2. Upload the fake score to your assignment server. Through your code.
3. add more weapons, assets are available in asset folder. Bind to other keys.
4. Add more projectiles like grenade. Bind to other keys.
5. Pic up specific ammo or weapon from map.
6. Add health to player. Assets are available.
7. customize your map to something else but a simple rectangle. Create your map on paper then add the points to setPoints.
8. Make map scrollable so you can have larger map that extends the frame. Possible can use ScrollPane.
9. Add another local player.
10. Add remote player.

Customization of graphics as new better colors or new images will fall under creative bonus everyone should be able to easily attempt this.