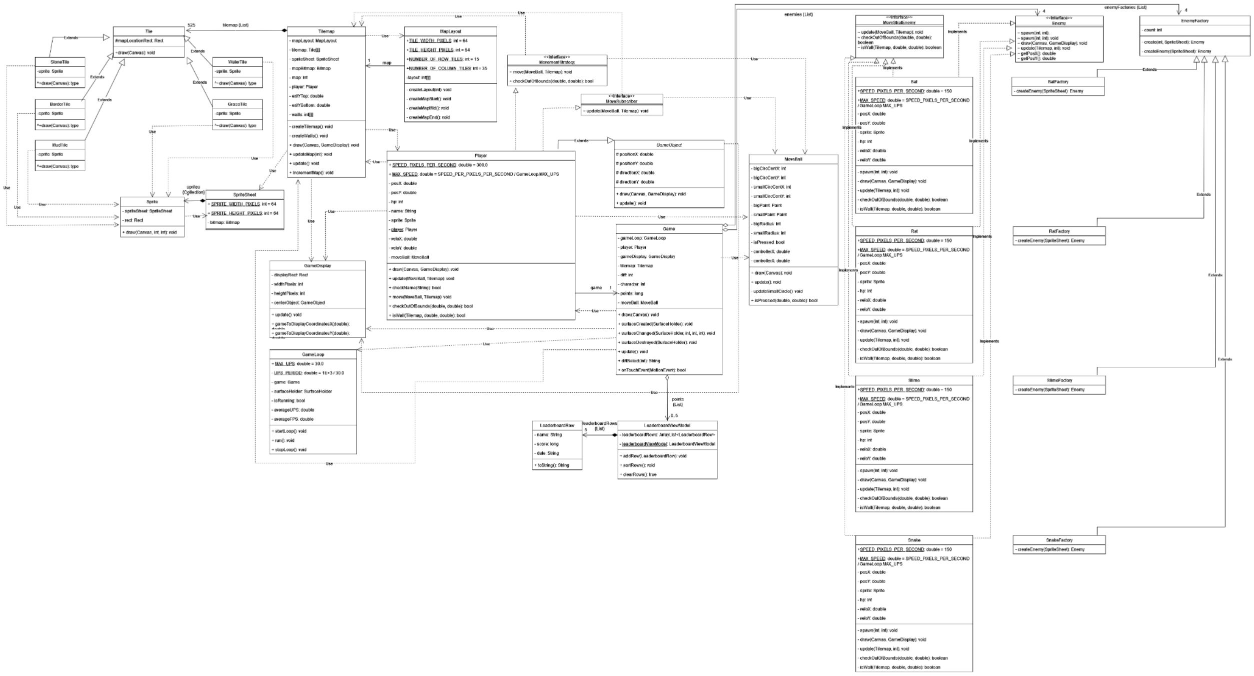
Revised Design Class Diagram

https://drive.google.com/file/d/1J6F0jZwvn1sYn7Js8jaCUPVwFJGYIN9W/view?usp=sharing


Tile
#mapLocationRect: Rect
+draw(Canvas): void
StoneTile
-sprite: Sprite
*+draw(Canvas): type
BorderTile
-sprite: Sprite
*+draw(Canvas): type
MudTile
-sprite: Sprite
*+draw(Canvas): type
WaterTile
-sprite: Sprite
^+draw(Canvas): type
GrassTile
-sprite: Sprite
^+draw(Canvas): type
Extends
Use
525
tilemap (List)
Tilemap
- mapLayout: MapLayout
- tilemap: Tile[][]
- spriteSheet: SpriteSheet
- mapBitmap: Bitmap
- map: int
- player: Player
- exitYTop: double
- exitYBottom: double
- walls: int[][]
- createTilemap(): void
- createWalls(): void
+ draw(Canvas, GameDisplay): void
+ updateMap(int): void
+ update(): void
+ incrementMap(): void
MapLayout
+TILE_WIDTH_PIXELS: int = 64
+TILE_HEIGHT_PIXELS: int = 64
+NUMBER_OF_ROW_TILES: int = 15
+NUMBER_OF_COLUMN_TILES: int = 35
-layout: int[][]
- createLayout(int): void
- createMapStart(): void
- createMapMid(): void
- createMapEnd(): void
1
map
Sprite
- spriteSheet: SpriteSheet
- rect: Rect
+ draw(Canvas, int, int): void
sprites (Collection)
SpriteSheet
+ SPRITE_WIDTH_PIXELS: int = 64
+ SPRITE_HEIGHT_PIXELS: int = 64
- bitmap: Bitmap
<<interface>>
MovementStrategy
~ move(MoveBall, Tilemap): void
~ checkOutOfBounds(double, double): bool
<<interface>>
MoveSubscriber
~ update(MoveBall, Tilemap): void
Player
+ SPEED_PIXELS_PER_SECOND: double = 300.0
+ MAX_SPEED: double = SPEED_PER_PIXELS_PER_SECOND / GameLoop.MAX_UPS
- posX: double
- posY: double
- hp: int
- name: String
- sprite: Sprite
- player: Player
- veloX: double
- veloY: double
- moveBall: MoveBall
+ draw(Canvas, GameDisplay): void
+ update(MoveBall, Tilemap): void
+ checkName(String): bool
+ move(MoveBall, Tilemap): void
+ checkOutOfBounds(double, double): bool
+ isWall(Tilemap, double, double): bool
GameObject
# positionX: double
# positionY: double
# directionX: double
# directionY: double
+ draw(Canvas, GameDisplay): void
+ update(): void
GameDisplay
- displayRect: Rect
- widthPixels: int
- heightPixels: int
- centerObject: GameObject
+ update(): void
+ gameToDisplayCoordinatesX(double): double
+ gameToDisplayCoordinatesY(double): double
GameLoop
+ MAX_UPS: double = 30.0
- UPS_PERIOD: double = 1E+3 / 30.0
- game: Game
- surfaceHolder: SurfaceHolder
- isRunning: bool
- averageUPS: double
- averageFPS: double
+ startLoop(): void
+ run(): void
+ stopLoop(): void
Game
- gameLoop: GameLoop
- player: Player
- gameDisplay: GameDisplay
- tilemap: Tilemap
- diff: int
- character: int
- points: long
- moveBall: MoveBall
+ draw(Canvas): void
+ surfaceCreated(SurfaceHolder): void
+ surfaceChanged(SurfaceHolder, int, int, int): void
+ surfaceDestroyed(SurfaceHolder): void
+ update(): void
+ diffSelect(int): String
+ onTouchEvent(MotionEvent): bool
game 1
points (List)
0..5
LeaderboardRow
- name: String
- score: long
- date: String
+ toString(): String
leaderboardRows (List)
5
LeaderboardViewModel
- leaderboardRows: ArrayList<LeaderboardRow>
- leaderboardViewModel: LeaderboardViewModel
+ addRow(LeaderboardRow): void
+ sortRows(): void
+ clearRows(): true
MoveBall
- bigCircCentX: int
- bigCircCentY: int
- smallCircCentX: int
- smallCircCentY: int
- bigPaint: Paint
- smallPaint: Paint
- bigRadius: int
- smallRadius: int
- isPressed: bool
- controllerX: double
- controllerX: double
+ draw(Canvas): void
+ update(): void
- updateSmallCircle(): void
+ isPressed(double, double): bool
enemies (List)
enemyFactories (List)
4
<<interface>>
MoveStratEnemy
~ update(MoveBall, Tilemap): void
~ checkOutOfBounds(double, double): boolean
~ isWall(Tilemap, double, double): boolean
<<interface>>
Enemy
~ spawn(int, int):
~ spawn(int, int): void
~ draw(Canvas, GameDisplay): void
~ update(Tilemap, int): void
~ getPosX(): double
~ getPosY(): double
Implements
EnemyFactory
- count: int
- create(int, SpriteSheet): Enemy
- createEnemy(SpriteSheet): Enemy
Bat
+SPEED_PIXELS_PER_SECOND: double = 150
+MAX_SPEED: double = SPEED_PIXELS_PER_SECOND / GameLoop.MAX_UPS
- posX: double
- posY: double
- sprite: Sprite
- hp: int
- veloX: double
- veloY: double
- spawn(int, int): void
- draw(Canvas, GameDisplay): void
- update(Tilemap, int): void
- checkOutOfBounds(double, double): boolean
- isWall(Tilemap, double, double): boolean
BatFactory
- createEnemy(SpriteSheet): Enemy
Rat
+SPEED_PIXELS_PER_SECOND: double = 150
+MAX_SPEED: double = SPEED_PIXELS_PER_SECOND / GameLoop.MAX_UPS
- posX: double
- posY: double
- sprite: Sprite
- hp: int
- veloX: double
- veloY: double
- spawn(int, int): void
- draw(Canvas, GameDisplay): void
- update(Tilemap, int): void
- checkOutOfBounds(double, double): boolean
- isWall(Tilemap, double, double): boolean
RatFactory
- createEnemy(SpriteSheet): Enemy
Slime
+SPEED_PIXELS_PER_SECOND: double = 150
+MAX_SPEED: double = SPEED_PIXELS_PER_SECOND / GameLoop.MAX_UPS
- posX: double
- posY: double
- sprite: Sprite
- hp: int
- veloX: double
- veloY: double
- spawn(int, int): void
- draw(Canvas, GameDisplay): void
- update(Tilemap, int): void
- checkOutOfBounds(double, double): boolean
- isWall(Tilemap, double, double): boolean
SlimeFactory
- createEnemy(SpriteSheet): Enemy
Snake
+SPEED_PIXELS_PER_SECOND: double = 150
+MAX_SPEED: double = SPEED_PIXELS_PER_SECOND / GameLoop.MAX_UPS
- posX: double
- posY: double
- sprite: Sprite
- hp: int
- veloX: double
- veloY: double
- spawn(int, int): void
- draw(Canvas, GameDisplay): void
- update(Tilemap, int): void
- checkOutOfBounds(double, double): boolean
- isWall(Tilemap, double, double): boolean
SnakeFactory
- createEnemy(SpriteSheet): Enemy

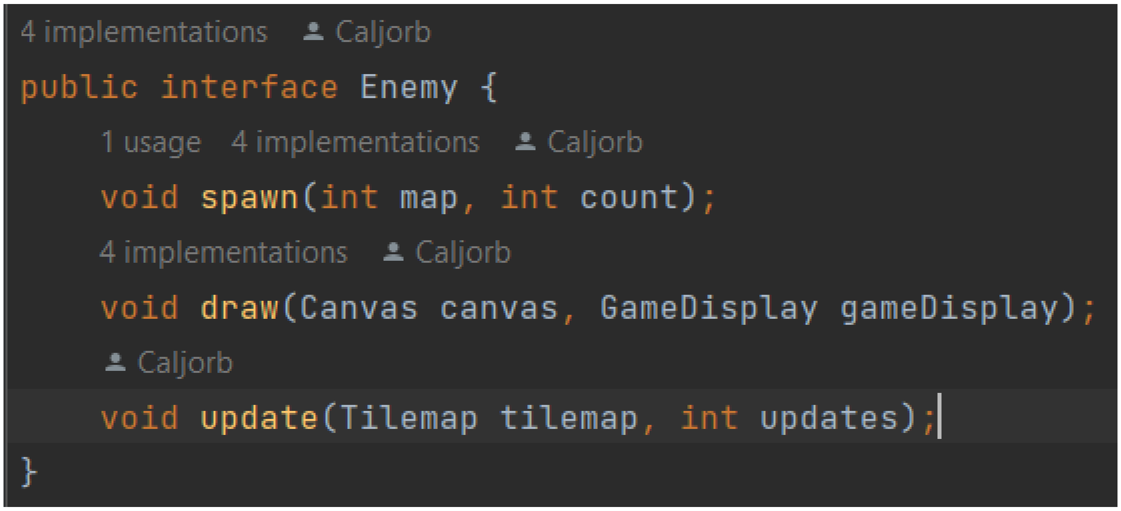
Link to diagram

ALL ELEMENTS TO THE RIGHT OF MOVEBALL ARE NEW

Design Pattern Evidence

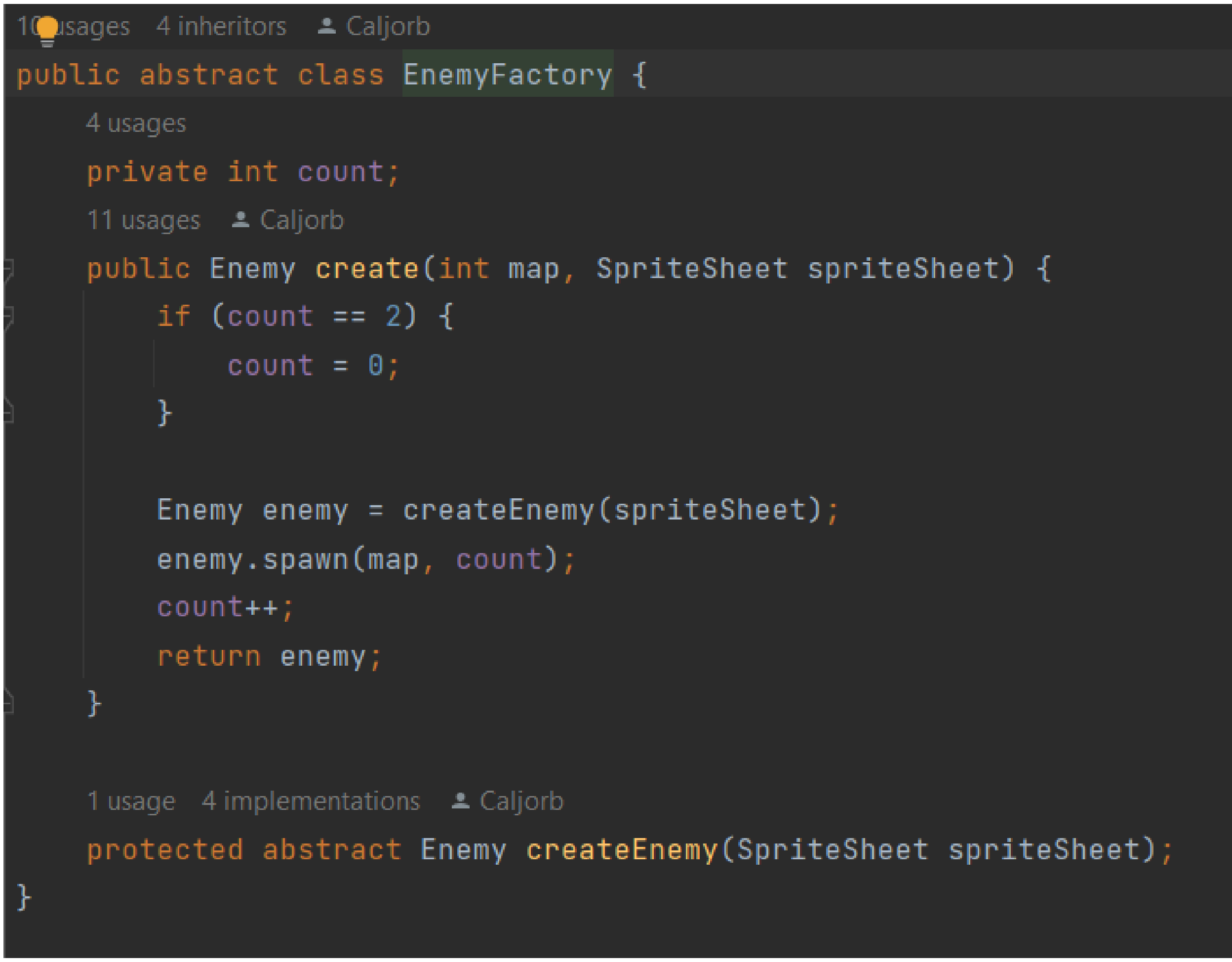
1. Factory Screenshots

    a.



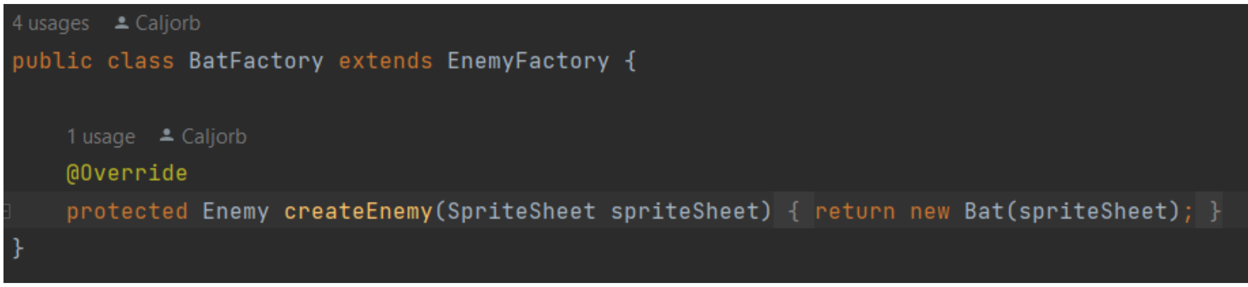
```
4 implementations   Caljorb
public interface Enemy {
    1 usage   4 implementations   Caljorb
    void spawn(int map, int count);
    4 implementations   Caljorb
    void draw(Canvas canvas, GameDisplay gameDisplay);
    Caljorb
    void update(Tilemap tilemap, int updates);
}
```

b.



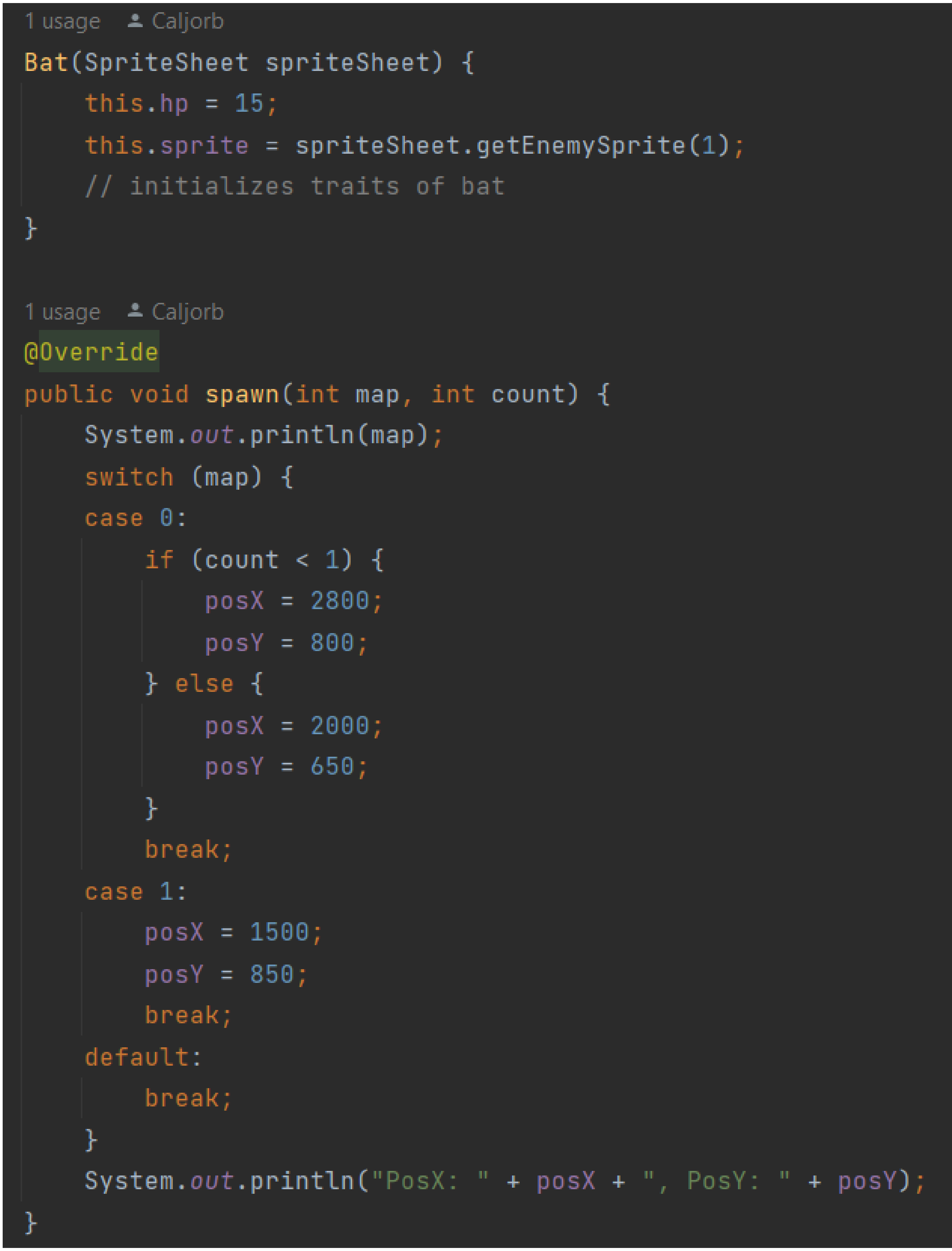
```
10 usages   4 inheritors   Caljorb
public abstract class EnemyFactory {
    4 usages
    private int count;
    11 usages   Caljorb
    public Enemy create(int map, SpriteSheet spriteSheet) {
        if (count == 2) {
            count = 0;
        }

        Enemy enemy = createEnemy(spriteSheet);
        enemy.spawn(map, count);
        count++;
        return enemy;
    }

    1 usage   4 implementations   Caljorb
    protected abstract Enemy createEnemy(SpriteSheet spriteSheet);
}
```

c.

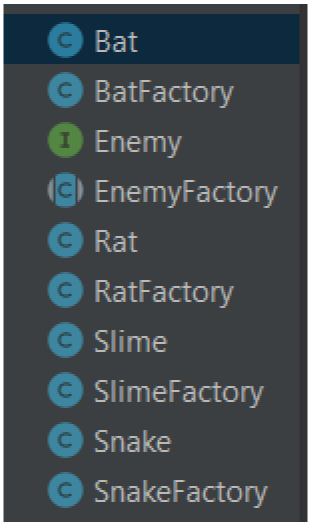


```
4 usages   Caljorb
public class BatFactory extends EnemyFactory {

    1 usage   Caljorb
    @Override
    protected Enemy createEnemy(SpriteSheet spriteSheet) { return new Bat(spriteSheet); }
}
```

d.

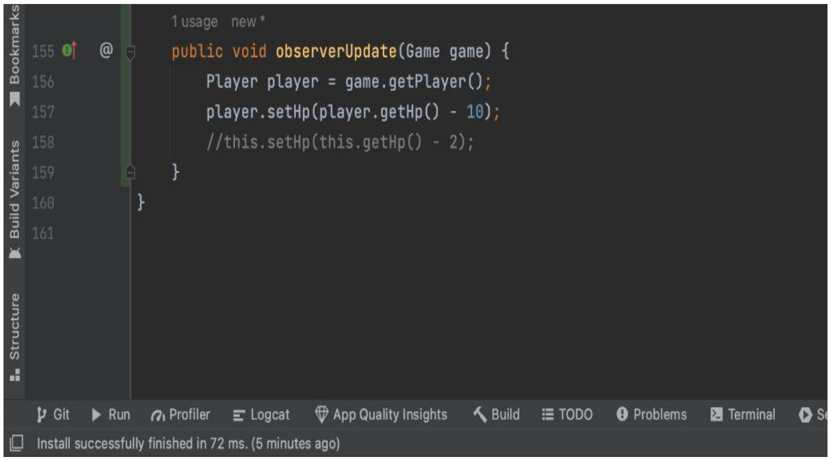


```
1 usage   Caljorb
Bat(SpriteSheet spriteSheet) {
    this.hp = 15;
    this.sprite = spriteSheet.getEnemySprite(1);
    // initializes traits of bat
}

1 usage   Caljorb
@Override
public void spawn(int map, int count) {
    System.out.println(map);
    switch (map) {
    case 0:
        if (count < 1) {
            posX = 2800;
            posY = 800;
        } else {
            posX = 2000;
            posY = 650;
        }
        break;
    case 1:
        posX = 1500;
        posY = 850;
        break;
    default:
        break;
    }
    System.out.println("PosX: " + posX + ", PosY: " + posY);
}
```

e. 

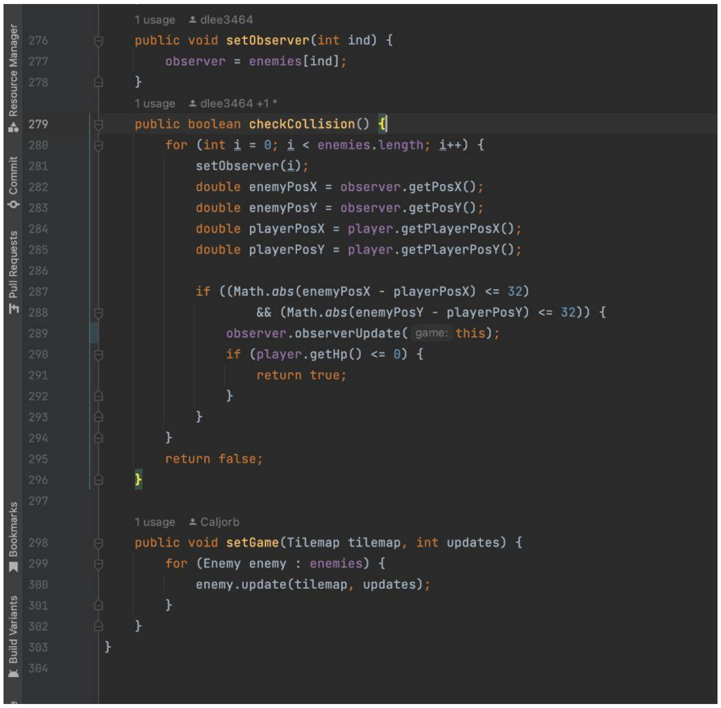
Bat
BatFactory
Enemy
EnemyFactory
Rat
RatFactory
Slime
SlimeFactory
Snake
SnakeFactory

2. Observer Screenshots

a. 

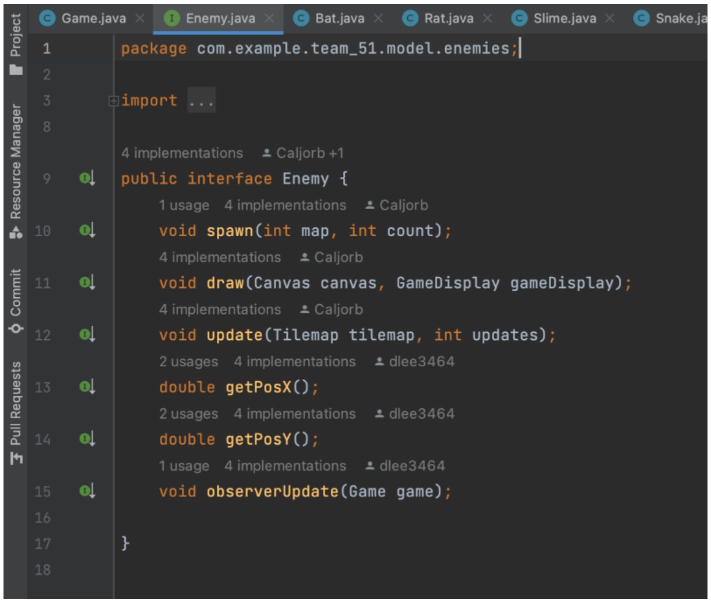
```
    public void observerUpdate(Game game) {
        Player player = game.getPlayer();
        player.setHp(player.getHp() - 10);
        //this.setHp(this.getHp() - 2);
    }
}
```

Install successfully finished in 72 ms. (5 minutes ago) b.



```
    1 usage   dlee3464
    public void setObserver(int ind) {
        observer = enemies[ind];
    }
    1 usage   dlee3464 +1 *
    public boolean checkCollision() {
        for (int i = 0; i < enemies.length; i++) {
            setObserver(i);
            double enemyPosX = observer.getPosX();
            double enemyPosY = observer.getPosY();
            double playerPosX = player.getPlayerPosX();
            double playerPosY = player.getPlayerPosY();

            if ((Math.abs(enemyPosX - playerPosX) <= 32)
                    && (Math.abs(enemyPosY - playerPosY) <= 32)) {
                observer.observerUpdate( game: this);
                if (player.getHp() <= 0) {
                    return true;
                }
            }
        }
        return false;
    }

    1 usage   Caljorb
    public void setGame(Tilemap tilemap, int updates) {
        for (Enemy enemy : enemies) {
            enemy.update(tilemap, updates);
        }
    }
}
```

c.



```
package com.example.team_51.model.enemies;

import ...

4 implementations   Caljorb +1
public interface Enemy {
    1 usage   4 implementations   Caljorb
    void spawn(int map, int count);
    4 implementations   Caljorb
    void draw(Canvas canvas, GameDisplay gameDisplay);
    4 implementations   Caljorb
    void update(Tilemap tilemap, int updates);
    2 usages   4 implementations   dlee3464
    double getPosX();
    2 usages   4 implementations   dlee3464
    double getPosY();
    1 usage   4 implementations   dlee3464
    void observerUpdate(Game game);

}
```

3. Factory Paragraph
    a. Our code follows the Factory pattern because we created an EnemyFactory that has the function of producing Enemies. This EnemFactory is an abstract class and gets implemented in each of the specific factories (concrete factories). There is also an Enemy interface that represents the “product” of an enemy, and then Bat, Slime, Snake, and Rat all implement this interface. In the Game class, we created the specific factories in

order to “produce” the enemies. Upon switching game screens, the factories produce different enemies as well. Finally, each factory has the create method, which first creates an enemy, then uses the spawn method from Enemy to give it a location. This allows for the inherit traits to always be the same for each enemy, while providing variation in where the enemies start on the screens through the spawn method, as it can put the enemies in different locations.

4. Observer Paragraph
    a. The observer pattern is a subscription-based pattern, in which events issued by a “publisher” can be observed by multiple “subscribers”. In our case, the Game class acts as the publisher that updates the player and enemy position, and the Enemy class is the subscriber interface. There are several different types of enemy classes like Bat, Snake, and Slime, which all implement the Enemy class and act as subscribers. The checkCollision() method in the Game class acts as the notifier if the conditions of collision are met. Inside the notifier method, the Enemy observer calls its observerUpdate method, which updates the player’s hp once the Enemy attacks it. The purpose of this design pattern is for the publisher to only have to access one Enemy interface when checking to notify each specific enemy observer.