**Ben Clark**

**Marcus Mertilien**

**December 21, 2017**

**Prof. A. Souza**

**CSC 413 – Section 02**

**A close up of a sign

Description generated with very high confidenceA close up of a sunset

Description generated with high confidence**

**Introduction**

At the onset of this project, my partner and I took into account the sum of our programming abilities as well as our ambitions and weighed those against the time constraints we were under. Considering the amount of time available, we chose to focus our energies primarily on the first game and design a highly portable architecture that would scale efficiently to our second game, Arkanoid which is “Super Rainbow Reef”. This is why later in the documentation you will see a large correlation between the classes used to build the Tank Wars, and those used to build Arkanoid. The purpose of this project was to expose students to large program development, as well as working in a group. The usage of GitHub as a version control system was a key element to successfully building from the same codebase between partners.

**Scope of Work**

* Tank Wars
  + Requirements:
    - Smooth Performance
    - Score Board
    - Background Music
    - Sound FX
    - Tank Angle & Moving Direction Handling
    - Mini-Map
    - Collisions
    - Split Screen Viewing Windows
    - Explosions FX
    - Power-Ups
  + Extras:
    - Independent Resource Acquisition
* Arkanoid
  + Requirements:
    - Smooth Performance
    - Score Board
    - Background Music
    - Sound FX
    - Ball to Paddle Angular Movement
    - Collisions
    - Level Transitions
    - Enemies
    - Explosions
  + Extras:
    - Independent Resource Acquisition
    - Power-Up Animations

**How To Play Tank Wars**

**Objective**:

The objective of the game is to use the boulders as cover to hide from your enemy while at the same time trying to destroy your enemy. But be careful, some boulders are not permanent coverage.

**Player Controls:**

**Player 1:**

Arrow Up: Move Upwards

Arrow Down: Move Downwards

Arrow Left: Move Left

Arrow Right: Move Right

Enter: Shoot

**Player 2:**

T: Move Upwards

G: Move Downwards

F: Move Left

H: Move Right

Space: Shoot

**Power Ups:**

|  |  |
| --- | --- |
|  | Increased Rate of Fire by 5fps (frames per second) |
| A picture containing clipart  Description generated with high confidence | Life +1 |
| A picture containing clipart  Description generated with high confidence | Increase Tank Movement by 1; |

**How To Play Arkanoid:**

**Objective:**

The objective of Arkanoid is to clear the stage of all the breakable bricks. You’ll find some of the power ups are helpful, and others hurt more than they help.

**Player Controls:**

**Player 1:**

Arrow Left: Move Left

Arrow Right: Move Right

Enter: Shoot

**Power Ups:**

|  |  |
| --- | --- |
|  | Enables Photon Lazers that destroy bricks when fired. |
|  | Temporarily cripples your thrusters and slows the ship. Careful more than one will add up. |
|  | Increases lives by 1. |
|  | Clears ALL power ups |

**Resources:**

* + **Tank Wars:**
    - Sprites: Sprites were acquired from Spriters-Resource.com
      * Credit for Ripping goes to the following:
        + SuperArthurBros ripped from “Super Mario All-Stars: Super Mario Bros. 3”
        + Death Egg ripped from “Gunstar Heroes”
        + Arima ripped from “Jackal / Top Gunner”
    - Sounds:
      * Explosion: FreeSoundEffects.com
      * Music: Provided
  + **Arkanoid:**
    - Title Screen:
      * JLukas ripped from SNES Arkanoid
    - Sprites: Sprites were acquired from Spriters-Resource.com
      * Credit for Ripping goes to the following:
        + Superjustinbros ripped from “Arkanoid”
    - Sounds:
      * Arkanoid SFX by J-Sinn ripped from “Arkanoid”

**Assumptions:**

* As a team we took the liberty of assuming that the second game would have to be completed within a week. With this being the case we focused heavily on building a solid and portable architecture in the first game in order to port it over to the second game. This development style allowed for a quick build of Arkanoid in about a weeks’ time frame.
* We assumed building the original game Arkanoid, instead of “Super Rainbow Reef” would be an acceptable substitution as a second game choice.

A close up of a map

Description generated with high confidence **Tank Wars Class Diagram**A close up of a map

Description generated with high confidence**Arkanoid Class Diagram**

**Tank Game Classes**

**Fields inherited from GameObject**

static int nextID, int \_id, int x, int y, int prevX, int prevY, int xSpeed, int ySpeed, int speed, boolean isVisible, boolean isSolid, int height, int width

**Constructor Summary**

public Actor(int x, int y, int width, int height, int xSpeed, int ySpeed, int speed, int direction)

- Construct a new Actor at the x,y position with the provided direction and speed properties

**Methods Summary**

public void draw(Graphics2D g2d)

- Draws the Actor using the provided Graphics2D object.

private void \_debugDraw(Graphics2D g2d)

- Private method that draws a white debug square around the Actor

private void \_draw(Graphics2D g2d)

- Private method that draws the Actor's sprite at its current position

**A screenshot of a social media post

Description generated with very high confidence**

**AudioTrack class**

public class AudioTrack

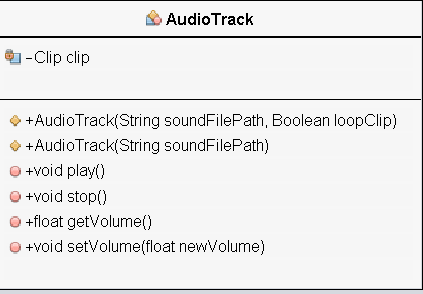
An AudioTrack is any SFX or music track playing within the game. The class possesses play, stop, and pause functionality along with volume control. AudioTracks are managed by the SoundManager class.

An AudioTrack knows how to play a .wav file at a provided file path

An AudioTrack knows how to play and stop its clip

An AudioTrack knows how to get and set its volume

An AudioTrack knows how to loop its clip.

**Fields**

private Clip clip - the audio clip associated with the class

**Constructor Summary**

public AudioTrack(String soundFilePath, Boolean loopClip)

- Create an AudioTrack using the file path and loop accordingly

public AudioTrack(String soundFilePath)

- Create an AudioTrack using the file path.

**Methods Summary**

public void play()

- Play the current clip

public void stop()

- Stop the current Clip

public float getVolume()

- Return the current clip volume

public void setVolume(float newVolume)

- Set the clip's volume

**Boulder class**

public class Boulder extends Prop

A Boulder is a type of static in-game object and as such extends Prop. Boulders come in two types - breakable, non-breakable - and the class contains a public enum containing these types for ensuring consistent typing while building objects of the class. Boulder contains a static initializer to create the two types' assets to be shared across all instances of the class.

A Boulder can be breakable or non-breakable (default).

A Boulder's has static assets shared between instances.

**Fields**

public Type type - The type of Boulder

public enum Type - The Types of Boulder (BREAKABLE, AND DEFAULT)

private static BufferedImage defaultBoulder - The asset for the default boulder type

private static BufferedImage breakableBoulder - The asset for the breakable boulder type

**Fields inherited from Prop**

static int nextID, int \_id, int x, int y, int prevX, int prevY, int xSpeed, int ySpeed, int speed, boolean isVisible, boolean isSolid, int height, int width

**Constructor Summary**

public Boulder(int x, int y, Type boulderType)

- Creates a new boulder of the provided type at the x, y coordinates.

**Methods Summary**

None

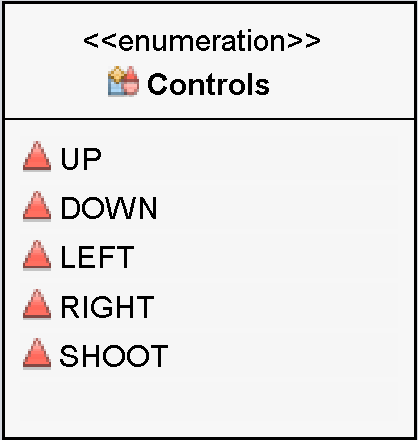
A screenshot of a cell phone

Description generated with high confidence

**Controls enum**

public enum Controls

The control enum represents the possible user inputs. The values are UP, DOWN, LEFT, RIGHT, and SHOOT. This enum is used in our input binding process - both in the GameEngine's setup sequence, and the user's Tank instance. By using this enum, we can create a hard binding between a KeyEvent constant - representing a key press - and a desired user movement (ex: KeyEvent.VK\_F => Controls.LEFT). This pattern allows for multiple players to be added without manually binding KeyEvents in the EventManager's Observers.

****

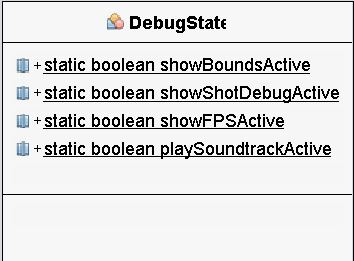
**DebugState class**

public class DebugState

The DebugState class is a small static instance that carries flags for toggling various debug behavior across the application. FSP, Collision, and audio features can be toggled during development.

The DebugState is capable of triggering debugging behavior within the application.

**Fields**

public static boolean showBoundsActive

- Triggers debug draw in our Actor and Prop classes to show collision boxes.

public static boolean showShotDebugActive

- Triggers the Projectile class to print debug data to the console.

public static boolean showFPSActive

- Triggers the GameEngine to print an FPS counter and stats to the console.

public static boolean playSoundtrackActive

- Triggers the SoundManager to stop playing that god awful chip tunes track...

**Constructor Summary – None Methods Summary - None**

**EventManager class**

public class EventManager extends Observable

Observable is a member of the JDK

The EventManager class is a small singleton wrapper that extends Observable for use with the InputHandler that is attached to the Game Engine’s JPanel. The EventManager provides an interface for the InputHandler to call when users interact with the keyboard. Players' Tank objects listen to the EventManager to trigger movement and other user interactions. This class also acts as an intercept point for other actions in the application like pause and exit.

The EventManager knows how to emit events that are monitored by its Observers.

**Fields**

private static EventManager instance - the singleton instance of the class.

A screenshot of a cell phone

Description generated with very high confidence

**Constructor Summary**

protected EventManager( )

- Block instantiation by classes outside of the current package.

**Methods Summary**

public static EventManager getInstance()

- Returns the class' singleton instance - or create one.

public void keyPressed(KeyEvent e)

- Trigged by the InputHandler on key press.

public void keyReleased(KeyEvent e)

- Trigged by the InputHandler on key released.

public void keyTyped(KeyEvent e)

- Trigged by the InputHandler on key typed.

**Explosion Class**

public class Explosion extends GameObject

The Explosion class is an SFX used by the GameEngine to show when something has been hit by a Projectile. It contains a static class initializer to gather and store the 8 frames for the explosion animation. When an Explosion is created, a timer counts down and dictates what animation frame should be displayed at a given moment. When the explosion animation has ended, the class marks itself as invisible to signal it is ready for the GameEngine's cleanup process.

An Explosion knows how long its animation is supposed to last.

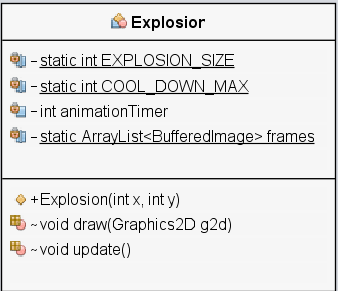
An Explosion knows how to draw the correct frame at the correct time.

An Explosion knows to hide itself when its animation has completed

**Fields**

private static int EXPLOSION\_SIZE - the explosion's size.

private static int COOL\_DOWN\_MAX - the initial cooldown amount of the explosion.

private int animationTimer - the instance’s current animation timer value.

private static ArrayList<BufferedImage> frames - frames that make up the animation.

**Constructor Summary**

public Explosion(int x, int y)

- Add a new Explosion at the provided coordinates.

**Methods Summary**

void draw(Graphics2D g2d)

- Draw the current animation frame.

void update()

- Update the animation timer one tick.

**GameEngine Class**

public class GameEngine extends JPanel implements Runnable

The GameEngine class is the core of the tank application. It manages everything from the application JFrame and FPS, to the player's key bindings and data layer. When executed, the class sets up all data, image assets, key bindings, and audio for the game before kicking off the game loop. The GameEngine will remain in the game loop until the application is terminated.

**Fields**

+ Game world size.

public static final int TILE\_SIZE

public static final int WORLD\_Y\_TILE\_COUNT

public static final int WORLD\_X\_TILE\_COUNT

public static final int WORLD\_WIDTH

public static final int WORLD\_HEIGHT

+ Player screen size.

private static final int VIEW\_SIZE

private static final int VIEW\_WIDTH

private static final int VIEW\_HEIGHT

+ View window size.

private static final int WINDOW\_BORDER\_WIDTH

private static final int WINDOW\_WIDTH

private static final int WINDOW\_HEIGHT

private static final Rectangle BOUNDS

+ Game loop constants.

private static final int TARGET\_FPS

private static final long ONE\_SECOND\_NS

private static final long OPTIMAL\_TIME

+ Assets

public static String ASSET\_PATH

public static String TANK\_ASSET\_PATH

public static String ENV\_ASSET\_PATH

public static String SOUND\_ASSET\_PATH

public static String EXPLOSION\_PATH

+ Game state

private static enum GameState

private Boolean isRunning

private GameState gameState

+ Managers

private EventManager eventManager

private InputHandler inputHandler

private SoundManager soundManager

+ Players

private static Tank player1

private static Tank player2

private HashMap<Integer, Controls> p1Keys

private HashMap<Integer, Controls> p2Keys

+ Data collections

private ArrayList<Tank> players

private ArrayList<Projectile> projectiles

private ArrayList<Boulder> boulders

private ArrayList<Projectile> player1Shots

private ArrayList<Projectile> player2Shots

private ArrayList<Explosion> explosions

+ Back buffer

private BufferedImage backgroundBuffer

**Constructor Summary**

None - uses default

**Method Summary**

public static void main(String[] args)

- Application entry point.

public void run( )

- Implemented for runnable.

public void init(JFrame frame)

- Initialize the application and add onto the provided JFrame.

private void setupKeys()

- Creates the key bindings both players.

private void setupBackground()

- Creates the background for the game world.

private void setupData()

- Creates and populates all data and collections for the application.

private void setupAudio()

- Sets up game audio.

private void gameLoop()

- Primary application loop. Once triggered, runs for the lifetime of the application.

private void updateData()

- Called from the game loop to update all in game objects by one tick.

private void checkCollisions()

- Called from the game loop to check for collisions between GameObjects in the game world

private void cleanupObjects()

- Called from the game loop to remove objects no longer visible.

@Override protected void paintComponent(Graphics g)

- Overrides JPanel's paintComponent function to be trigger on repaint() call in the game loop.

A screenshot of a cell phone

Description generated with very high confidenceprivate void drawViews(Graphics2D g2d, BufferedImage gameWorldBuffer)

- Called by the game loop to draw the each side of the players split screens.

private void drawBackground(Graphics2D g2d)

- Draws the background buffer to the provided Graphics2D object.

private void drawGameObjects(Graphics2D g2d)

- Triggers all GameObjects to be drawn to the provided Graphics2D object.

private void drawFXObjects(Graphics2D g2d)

- Triggers all Explosions to be drawn to the provided Graphics2D object.

private void drawMiniMaps(Graphics2D g)

- Draws the mini map UI for each split screen.

private boolean damage(Tank attacker, Tank defender)

- Applies damage from attacker to defender as a result of a Projectile to Tank collision.

A screenshot of a cell phone

Description generated with very high confidence

**GameObject class**

public abstract class GameObject

The base abstract class for all in-game objects. The class contains the basic information about the object: size, position, previous position, visibility, solidity, and sprite. As the draw needs of the child Prop and Actor classes are slightly different, an abstract draw class must be overridden to assure all child types know how to draw themselves.

A GameObject knows its own location.

A GameObject knows its own size.

A GameObject knows if its visible.

**A screenshot of a cell phone

Description generated with very high confidence**

**Fields**

static int nextId - Id for next built object

int \_id - Current object's id

int x - Horizontal position

int y - Vertical position

int prevX - Previous x position

int prevY - Previous y position

boolean isVisible - Dictates if the object is visible

boolean isSolid - Dictates if the object is solid

int height - Dictates height

int width - Dictates width

BufferedImage sprite - The image to be drawn each frame

**Constructor Summary**

public GameObject(int x, int y, int width, int height, boolean isSolid, boolean isVisible)

- Create a game object with the provide position, size, and state.

A screenshot of a cell phone

Description generated with very high confidence**Methods Summary**

abstract void draw(Graphics2D g2d)

- Abstract draw that must be implemented by child classes

public boolean isVisible()

- Get visibility state

public boolean isHidden()

- Get visibility state

public void hide()

- Make GameObject invisible

public void show()

- Make GameObject visible

public void resetLocation()

- Reset GameObject to previous x, y position

public Rectangle getBound()

- Get the bounds of the GameObject

public Point getLocation()

- Get the location of the GameObject

public Point getCenterLocation()

- Get the center location of the rectangle formed by the GameObject

**InputHandler class**

public class InputHandler implements KeyListener

The InputHandler is attached to the GameEngine's JPanel and handles all user input from the keyboard. In conjunction with the EventManager, key presses are relayed from the JPanel up to the Player's Tank instance where movement and shooting are triggered. This class is a singleton so only one instance is required regardless of how many players are currently in the game.

The InputHandler knows when a KeyEvent has been triggered

The InputHandler is responsible for relaying events to out event system

**Fields**

private static InputHandler instance - The reference to the singleton instance.

private static EventManager eventManager - A reference to the EventManager singleton.

A screenshot of a cell phone

Description generated with very high confidence**Constructor Summary**

protected InputHandler()

- Block instantiation by classes outside of this package.

**Methods Summary**

public static InputHandler getInstance()

- Return the singleton instance

@Override public void keyTyped(KeyEvent e)

- Overrides KeyListener keyTyped

@Override public void keyPressed(KeyEvent e)

- Overrides KeyListener keyPressed

@Override public void keyReleased(KeyEvent e)

- Overrides KeyListener keyReleased

**Physics Class**

public class Physics

A small wrapper class to contain the collision and bounded checks for interactions of the GameObjects. The physics class allows you to check if GameObjects are contained by other objects(bounded), or intersect with other objects(collide).

The Physics class knows how to check if GameObjects collide or are contained by one another.

**Field Summary**

None

A screenshot of a cell phone

Description generated with very high confidence

**Constructors Summary**

None

**Method Summary**

public static Boolean collides(GameObject objA, GameObject objB)

- Checks if the two GameObjects collide

public static Boolean collides(GameObject obj, Rectangle bounds)

- Checks if the GameObject collides with the Rectangle

public static Boolean bounded(GameObject objA, GameObject objB)

- Checks if the GameObjectA is bounded by GameObjectB

public static Boolean bounded(GameObject obj, Rectangle bounds)

- Checks if the GameObject is bounded by the Rectangle

**Projectile class**

public class Projectile extends Actor

The Projectile class represents a bullet fired by the Tank class. When a Projectile is created it will travel in the same direction until hitting a Tank, an obstacle, or a boundary. The Projectile class uses a static initializer to load its image asset to be shared across all instances.

A Projectile can not change direction once fired.

A screenshot of a cell phone

Description generated with very high confidenceA Prop can draw itself.

A Prop knows to draw a debugged version of itself.

**Fields Summary**

public static int PROJECTILE\_SIZE - the size of the projectile

public static int PROJECTILE\_SPEED - the default speed of a Projectile

**Constructors**

public Projectile(int direction, int x, int y, int xSpeed, int ySpeed)

- Create a new Projectile with the provided position and speed properties

**Methods Summary**

public void update( )

- Update the projectile one game tick

**Prop class**

public abstract class Prop extends GameObject

A Prop represents an in game object that does not move and is only interacted with by Actors. The Prop only needs to know how to draw itself. Once placed, the Prop never moves until it is removed from the game world. Prop acts as parent class for any static item within the game world.

A Prop does not moveA screenshot of a cell phone

Description generated with very high confidence and can only be interacted with.

A Prop can draw itself, and also knows to draw a debugged version of itself.

**Fields Summary**

None

**Fields inherited from GameObject**

static int nextID, int \_id, int x, int y,

int prevX, int prevY, int xSpeed, int ySpeed,

int speed, boolean isVisible, boolean isSolid,

int height, int width

**Constructors Summary**

public Prop(int x, int y, int width, int height)

- Create a new Prop at the x, y with the provided height and width.

**Methods Summary**

@Override public void draw(Graphics2D g2d)

- Draw the Props sprite to the g2d instance

private void \_debugDraw(Graphics2D g2d)

- Draw a white collision box around the Prop

private void \_draw(Graphics2D g2d)

- Draw the Prop's sprite asset.

**SoundManager Class**

public class SoundManager

The SoundManager manages all audio actions within the game. It presents an API to play audio samples - shots or explosions - and tracks - background music. The class is a singleton instance that is shared between all objects in the application. The manager runs on the same thread the application.

The SoundManager knows how to play and stop SFX.

The SoundManager knows how to play and stop music.A screenshot of a cell phone

Description generated with very high confidence**Fields Summary**

private static SoundManager instance - The singleton instance reference

private AudioTrack soundtrack - The backing music for the game

**Constructors Summary**

protected SoundManager() { }

- Block instantiation by classes outside this package.

**Methods Summary**

public static SoundManager getInstance()

- Return the singleton instance

public void playSoundtrack()  
- Play the soundtrack

public void stopSoundtrack()

- Stop the soundtrack

public void playShot()

- Play the shot SFX

public void playExplosion()

- Play the Explosion SFX

**Tank Class**

public class Tank extends Actor implements Observer

A screenshot of a cell phone

Description generated with very high confidenceThe Tank class the primary player GameObject and interacts with almost all of the application during its lifespan. The class contains a static initializer to setup the image assets for both versions of the tank enabling them to be shared between all instances of the class. The class also contains a Colors enum required by the constructor to type ensure any potential new instances. To add more colors simply add a new type and image asset.

**Fields Summary**

private static final int TANK\_SIZE - the default width and height of the Tank

private static final int TANK\_SPEED - default movement speed of the Tank

private static final int FIRING\_SPEED - the default firing speed of the Tank

private static final int TOTAL\_HEALTH - default starting health of the Tank

private static final int TOTAL\_LIVES - the default total lives count of the Tank

public Colors color - the color of the current Tank instance

public enum Colors - enum for the possible Tank colors

private static BufferedImage redTank - an image asset for the green tank

private static BufferedImage greenTank - an image asset for the red tank

private HashMap<Controls, Boolean> buttonStates

- a HashMap to track which player buttons are currently pressed

private HashMap<Integer, Controls> controlMap;

- a HashMap indicating which keys the instance should be listening for

private int shotCooldown - a cooldown counter to prevent bullet spam

private int originX - spawn x position of the tank

private int originY - spawn Y position of the tank

protected int score - current score

protected int health - current health

protected int lives - current remaining lives

**Constructors Summary**

public Tank(int x, int y, int speed, int size, HashMap<Integer, Controls> controls, Colors color)

- Create a Tank with the provided position, speed, size, control map and color type.

**Methods Summary**

private void \_initControls(HashMap<Integer, Controls> controls)

- Map the controller setup to the instance's button states.

public void update(ArrayList<Projectile> shots)

- Update the Tank one in-game tick.

private void \_updatePosition()

- Update the x, y position of the Tank.

private void \_updateDirection()

- Update the direction of the Tank.

private void \_updateShoot(ArrayList<Projectile> shots)

- Update the Tank's Projectiles.

@Override public void update(Observable obj, Object e)

- Triggered by our event handler when a KeyEvent is present.

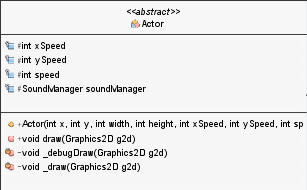
public void goHome()

- Return the Tank to its spawn x, y position.

**Arkanoid Classes**

**Arkanoid Actor class**

public abstract class Actor extends GameObject  
  
The Actor class is used in our inheritance hierarchy as the primary abstract class to represent moving objects. The Ship, Ball and Enemy classes extend Actor.



**Fields**

int xSpeed - the Actor’s horizontal speed

int ySpeed - the Actor’s vertical speed

int speed - the base speed of the Actor

SoundManager soundManager - access to application sound system

**Constructors**

public Actor(int x,int y,int width,int height,int xSpeed, int ySpeed, int speed)

**Methods**

public void draw(Graphics2D g2d)

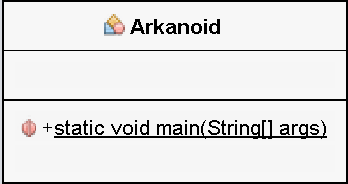
private void \_debugDraw(Graphics2D g2d)

private void \_draw(Graphics2D g2d)

**Arkanoid class**

public class Arkanoid

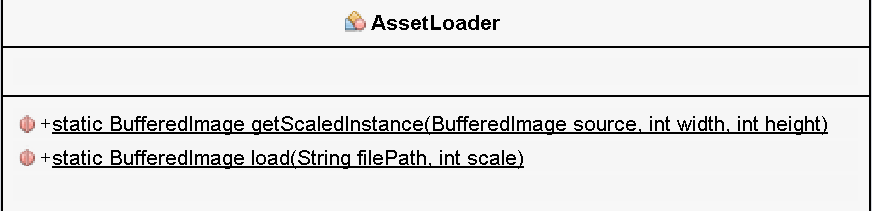
The Arkanoid class is the wrapper for the primary application and acts as an entry point for the game.



**Arkanoid AssetLoader class**

public class AssetLoader

The asset loader is a small static class that helps with the loading and scaling of image assets for the application.

**Fields**

None

**Constructors**

None

**Methods**

public static BufferedImage getScaledInstance(BufferedImage source, int width, int height)

* Return a new image at the provided size

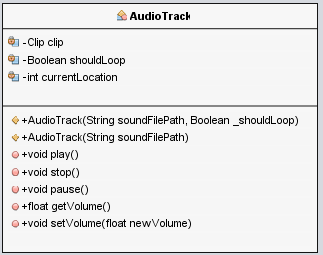
public static BufferedImage load(String filePath, int scale) {

* Load the specified file, at the specified scale.

**Arkanoid AudioTrack class**

public class AudioTrack

An AudioTrack represents an audio clip or piece of music within the application. It has the basic functionality for play, stop, pause, and volume control.  
  
**Fields**

private Clip clip - the audio clip to be played

private Boolean shouldLoop - whether the clip loops

private int currentLocation - the location of the current pause point

**Constructors**

public AudioTrack(String soundFilePath, Boolean \_shouldLoop)

public AudioTrack(String soundFilePath)

**Methods**

public void play()

public void stop()

public void pause()

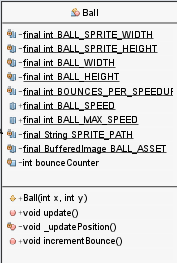
public float getVolume()

public void setVolume(float newVolume)

**Arkanoid Ball class**

public class Ball extends Actor

This class is responsible for tracking the ball in Arkanoid. Constants insure uniform drawing, and by extending Actor, Ball also become a gameObject. Actor also handles the drawing done in gameEngine.



**Constants**

private static final int BALL\_SPRITE\_WIDTH - sprite asset width

private static final int BALL\_SPRITE\_HEIGHT - sprite asset height

private static final int BALL\_WIDTH - visible ball width

private static final int BALL\_HEIGHT - visible ball height

private static final int BOUNCES\_PER\_SPEEDUP - speed up marker

public static final int BALL\_SPEED - default ball speed

public static final int BALL\_MAX\_SPEED - the max ball speed

private static final String SPRITE\_PATH - asset path

private static final BufferedImage - ball asset

**Fields**

private int bounceCounter - keep track of bound count for ball speed up

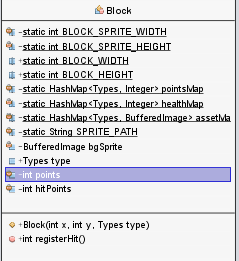
**Constructor**

public Ball(int x, int y)  
  
**Methods**

public void update()

private void \_updatePosition()

public void incrementBounce()

**Arkanoid Block class**

public class Block extends Prop

Blocks are the main obstacles of the game. Constants again insure uniform drawing. Extending Prop provides drawing methods for Blocks.

**Constants**

private static int BLOCK\_SPRITE\_WIDTH - width of stage area

private static int BLOCK\_SPRITE\_HEIGHT - height of stage area

public static int BLOCK\_WIDTH - visible block width

public static int BLOCK\_HEIGHT - visible block height

**Statics**

private static HashMap<Types, Integer> pointsMap - the map of all block point amounts

private static HashMap<Types, Integer> healthMap - the map of all block health amounts

private static HashMap<Types, BufferedImage> assetMap - the map of all block types

private static String SPRITE\_PATH - the sprite path

public static enum Types - WHITE, YELLOW, PINK, BLUE, RED, GREEN, CYAN, ORANGE, SILVER, GOLD

**Fields**

private BufferedImage bgSprite - the block asset

public Types type - the type of Block

private int points - the number of points the block is worth

private int hitPoints - the number of hit points the block has

**Constructors**

public Block(int x, int y, Types type)

**Methods**

public int registerHit( )

**Arkanoid Controls enum**

public enum Controls

A small enum class to help manage play movements within the game.  
  
**Values**

LEFT, RIGHT, SHOOT, PAUSE, EXIT, START

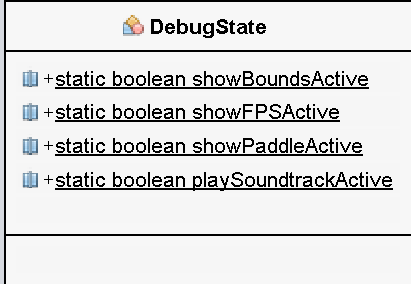
A screenshot of a cell phone

Description generated with very high confidence

**Arkanoid DebugState class**

public class DebugState

The DebugState class is a small static instance that carries flags for toggling various debug behavior across the application. FSP, Collision, and audio features can be toggled during development.

****

**Fields**

public static boolean showBoundsActive

public static boolean showFPSActive

public static boolean showPaddleActive

public static boolean playSoundtrackActive

**Constructor Summary**

None

**Methods Summary**

None

**Arkanoid Enemy class**

public class Enemy extends Prop

**Constants**

private static final int ENEMY\_WIDTH - width of enemy

private static final int ENEMY\_HEIGHT -  height of enemy

private static final int ANIMATION\_COOLDOWN - animation max time out

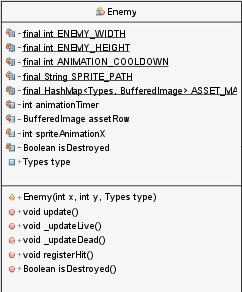
private static final String SPRITE\_PATH - asset path

private static final HashMap<Types, BufferedImage> ASSET\_MAP - the asset map for all enemy types

public static enum Types - possible types: GREEN, RED, BLUE

**Fields**

private int animationTimer - animation timer for roll effect

****private BufferedImage assetRow - the animation asset row

private int spriteAnimationX - the current x positon of the animation

private Boolean isDestroyed - f this is still a valid enemy

public Types type - the enemy's type

**Constructors**

public Enemy(int x, int y, Types type)

**Methods**

public void update()

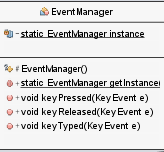
public void \_updateLive()

public void \_updateDead()

public void registerHit()

public Boolean isDestroyed()

**Arkanoid EventManager class**



public class EventManager extends Observable

**fields**

private static EventManager instance - the singleton instance

**Constructors**

protected EventManager()

**Methods**

public static EventManager getInstance()

public void keyPressed(KeyEvent e)

public void keyReleased(KeyEvent e)

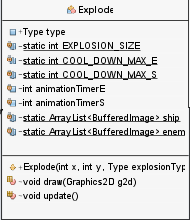
public void keyTyped(KeyEvent e)

**Arkanoid Explode class**

public class Explode extends GameObject

**Constants**

private static int EXPLOSION\_SIZE -

****private static int COOL\_DOWN\_MAX\_E -

private static int COOL\_DOWN\_MAX\_S -

private int animationTimerE -

private int animationTimerS -

public enum Type - possible types: SHIP,ENEMY

**Fields**

public Type type -

private static ArrayList<BufferedImage> ship -

private static ArrayList<BufferedImage> enemy -

**Constructors**

public Explode(int x, int y, Type explosionType)

**Methods**

void draw(Graphics2D g2d)

void update()

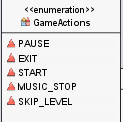
**Arkanoid GameActions enum**

public enum GameActions

A small enum class to help with player interactions with the underlying game engine. This class is key in creating an efficient and easy to use API for later implementations.

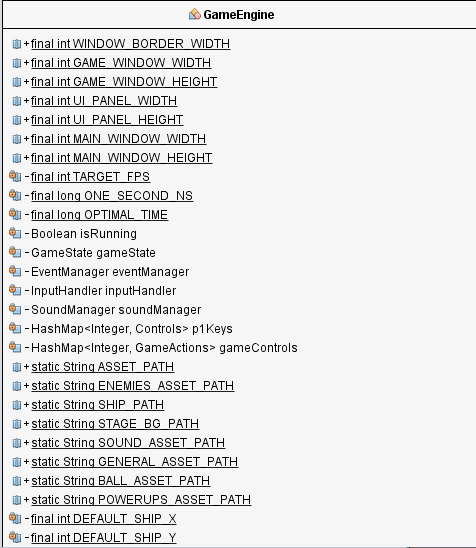
**Values**

PAUSE, EXIT, START, MUSIC\_STOP, SKIP\_LEVEL



**Arkanoid GameEngine class**

public class GameEngine extends JPanel implements Runnable, Observer



**Constants**

public static final int WINDOW\_BORDER\_WIDTH -

public static final int GAME\_WINDOW\_WIDTH -

public static final int GAME\_WINDOW\_HEIGHT -

public static final int UI\_PANEL\_WIDTH -

public static final int UI\_PANEL\_HEIGHT -

public static final int MAIN\_WINDOW\_WIDTH -

public static final int MAIN\_WINDOW\_HEIGHT -

private static final int TARGET\_FPS -

private static final long ONE\_SECOND\_NS -

private static final long OPTIMAL\_TIME -

public static String ASSET\_PATH -

public static String ENEMIES\_ASSET\_PATH -

public static String SHIP\_PATH -

public static String STAGE\_BG\_PATH -

public static String SOUND\_ASSET\_PATH -

public static String GENERAL\_ASSET\_PATH -

public static String BALL\_ASSET\_PATH -

public static String POWERUPS\_ASSET\_PATH -

private static final int DEFAULT\_SHIP\_X -

private static final int DEFAULT\_SHIP\_Y -

private static final int DEFAULT\_BALL\_X -

private static final int DEFAULT\_BALL\_Y -

private static enum GameState - possible values: MAIN\_MENU, GAME\_RUNNING, PAUSE\_MENU, ROUND\_CHANGE, PLAYER\_DIED, GAME\_OVER, GAME\_WON, EXITING

**Fields**

private Boolean isRunning –

private GameState gameState-

private EventManager eventManager -

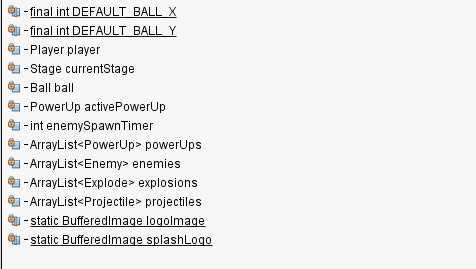
private InputHandler inputHandler -

private SoundManager soundManager -

private HashMap<Integer, Controls> p1Keys -

private HashMap<Integer, GameActions> gameControls -

private Player player -

****private Stage currentStage -

private Ball ball -

private PowerUp activePowerUp -

private int enemySpawnTimer -

private HashMap<String, Integer> scores -

private ArrayList<PowerUp> powerUps -

private ArrayList<Enemy> enemies -

private ArrayList<Explode> explosions -

private ArrayList<Projectile> projectiles -

private static BufferedImage logoImage -

private static BufferedImage splashLogo -

**Methods**

public void run()

public void init(JFrame frame)

rivate void \_setupControls()

private void \_setupGameData()

private void \_setupGameAudio()

private void gameLoop()

private void \_updateData()

private void \_updateEnemies()

private void \_checkCollisions()

private void \_cleanupObjects()

private void \_checkState()

private void \_resetPlayer()

private void \_resetStage()

protected void paintComponent(Graphics g)

private void \_drawGameWorld(Graphics2D g2d)

private void \_drawGameUI(Graphics2D g2d)

private void \_drawPauseMenu(Graphics2D g2d)

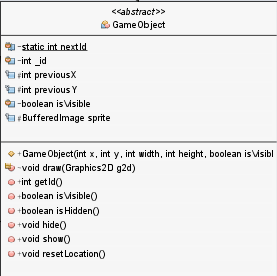
private void \_drawMainMenu(Graphics2D g2d)

private void \_drawSplashScreen(Graphics2D g2d, String title, String cta, Color titleColor)

public void update(Observable obj, Object e)

**Arkanoid GameObject class**

public abstract class GameObject extends Rectangle

**Fields**

private static int nextId - id for next object

private int \_id - current object id

protected int previousX - Previous x position

protected int previousY - Previous y position

private boolean isVisible - dictates if the object is visible

protected BufferedImage sprite - the image to be drawn each frame

**Constructors**

public GameObject(int x, int y, int width, int height, boolean isVisible)

**Methods**

abstract void draw(Graphics2D g2d)

public int getId()

public boolean isVisible()

public boolean isHidden()

public void hide()

public void show()

public void resetLocation()

**Arkanoid InputHandler class**

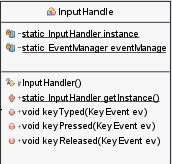
public class InputHandler implements KeyListener

**Fields**

private static InputHandler instance - The reference to the singleton instance.

private static EventManager eventManager - A reference to the EventManager singleton.

**Constructor Summary**

****protected InputHandler()

- Block instantiation by classes outside of this package.

**Methods Summary**

public static InputHandler getInstance()

- Return the singleton instance

public void keyTyped(KeyEvent e)

- Overrides KeyListener keyTyped

public void keyPressed(KeyEvent e)

- Overrides KeyListener keyPressed

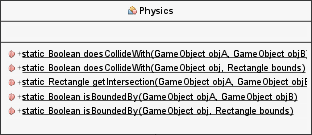
public void keyReleased(KeyEvent e)

- Overrides KeyListener keyReleased

**Arkanoid Physics class**

public class Physics

This class acts as a wrapper for the collision detection predefined in the Rectangle class. By creating a wrapper collision detection in the GameEngine class becomes greatly simplified. This class also takes advantage of inheritance by using GameObjects as parameters. This allows for flexibility throughout the code base.

****

**Fields**

None

**Constructor**

None

**Methods**

public static Boolean doesCollideWith(GameObject objA, GameObject objB)

public static Boolean doesCollideWith(GameObject obj, Rectangle bounds)

public static Rectangle getIntersection(GameObject objA, GameObject objB)

public static Boolean isBoundedBy(GameObject objA, GameObject objB)

public static Boolean isBoundedBy(GameObject obj, Rectangle bounds)

**Arkanoid Player class**

public class Player extends Ship implements Observer

**Constants**

private final static int FIRING\_SPEED -

private static final int DEFAULT\_LIVES -

private static final int DEFAULT\_SCORE -

**Fields**

private HashMap<Controls, Boolean> buttonStates - current pressed buttons

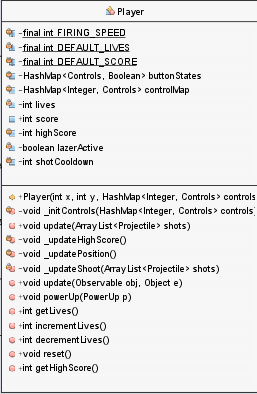
private HashMap<Integer, Controls> controlMap - map of keys to player controls

public int lives

public int score

private boolean lazerActive

private int shotCooldown



**Constructors**

public Player(int x, int y, HashMap<Integer, Controls> controls)

**Methods**

private void \_initControls(HashMap<Integer, Controls> controls) -

public void update(ArrayList<Projectile> shots) -

private void \_updatePosition() -

private void \_updateShoot(ArrayList<Projectile> shots) -

public void update(Observable obj, Object e) -

public void powerUp(PowerUp p) -

public int getLives() -

public int incrementLives() -

public int decrementLives() -

public void reset() -

**Arkanoid PowerUp class**

public class PowerUp extends Prop

**Constants**

public enum Types - possible values: LAZER, EXTEND, SLOW, CATCH, BREAK, SPEED\_UP, TWIN, NEWDISRUPT, PLAYER, REDUCE

private static final int POWERUP\_SPRITE\_WIDTH - width of power up

private static final int POWERUP\_SPRITE\_HEIGHT - height of power up

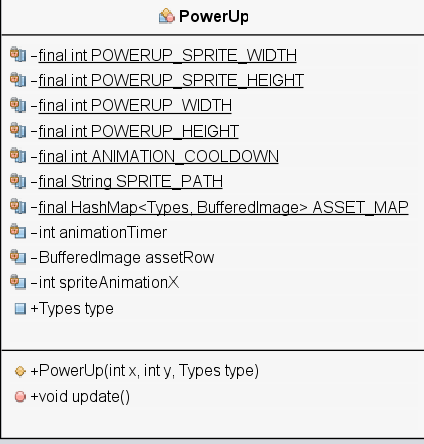
private static final int POWERUP\_WIDTH - visible width

private static final int POWERUP\_HEIGHT - visible height

private static final int ANIMATION\_COOLDOWN - animation max time out

private static final String SPRITE\_PATH - spriate path

private static final HashMap<Types, BufferedImage> ASSET\_MAP - static asset map



**Fields**

private int animationTimer - animation timer for roll effect

private BufferedImage assetRow - the animation asset row

private int spriteAnimationX - the current x positon of the animation

public Types type -the power up's type

**Constructors**

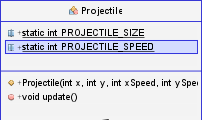
public PowerUp(int x, int y, Types type)

**Methods**

public void update()

**Arkanoid Projectile class**

public class Projectile extends Actor

****

**Constants**

public static int PROJECTILE\_SIZE -

public static int PROJECTILE\_SPEED -

**Constructors**

public Projectile(int x, int y, int xSpeed, int ySpeed)

**Methods**

public void update()

**Arkanoid Prop class**

public abstract class Prop extends GameObject

**Constructors**

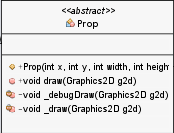
public Prop(int x, int y, int width, int height)

**Methods**

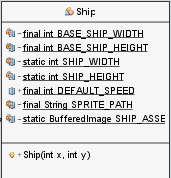
public void draw(Graphics2D g2d)

private void \_debugDraw(Graphics2D g2d)

private void \_draw(Graphics2D g2d)

****

**Arkanoid Ship class**

****public class Ship extends Actor

**Constants**

private static final int BASE\_SHIP\_WIDTH -

private static final int BASE\_SHIP\_HEIGHT -

private static int SHIP\_WIDTH -

private static int SHIP\_HEIGHT -

public static final int DEFAULT\_SPEED -

private static final String SPRITE\_PATH -

private static  BufferedImage SHIP\_ASSET -

**Constructors**

public Ship(int x, int y)

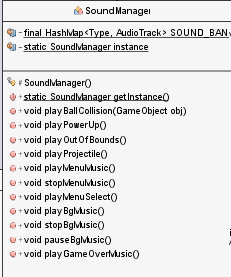
**Arkanoid SoundManager class**

public class SoundManager

**Constants**

private static enum Type - SFX type

private static final HashMap<Type, AudioTrack> SOUND\_BANK - sound storage

****

**Fields**

private static SoundManager instance

**Constructors**

protected SoundManager()

**Methods**

public static SoundManager getInstance()

public void playBallCollision(GameObject obj)

public void playPowerUp()

public void playOutOfBounds()

public void playProjectile()

public void playMenuMusic()

public void stopMenuMusic()

public void playMenuSelect()

public void playBgMusic()

public void stopBgMusic()

public void pauseBgMusic()

public void playGameOverMusic()

**Arkanoid Stage class**

public class Stage

**Constants**

private static final int BG\_SPRITE\_WIDTH - width of stage area

private static final int BG\_SPRITE\_HEIGHT - height of stage area

private static final int SPACER\_WIDTH - horizonatal spacer between stage bgs

private static final int SPACE\_HEIGHT - vertical spacer between bgs

private static final int STAGE\_WIDTH - visual stage width

private static final int STAGE\_HEIGHT - visual stage height

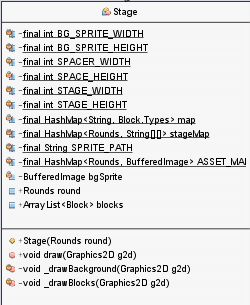
public static enum Rounds - possible rounds: ROUND\_1, ROUND\_2, ROUND\_3, ROUND\_4, ROUND\_5

private static final HashMap<String, Block.Types> map -

private static final HashMap<Rounds, String[][]> stageMap -

private static final String SPRITE\_PATH -

private static final HashMap<Rounds, BufferedImage> ASSET\_MAP -

**Fields**

private BufferedImage bgSprite

public Rounds round

public ArrayList<Block> blocks

**Constructors**

public Stage(Rounds round)

**Methods**

public void draw(Graphics2D g2d)

private void \_drawBackground(Graphics2D g2d)

private void \_drawBlocks(Graphics2D g2d)

**Arkanoid and Tank Wars Shared Classes**

* **Actor**
  + Acts as a wrapper for objects such as “Tank” and “Ship”. Actor handles the drawing and stores player constants such as object sizes, Firing speeds, initial Health, and initial lives.
* **Audio Track**
  + Creates a public API that allows the manipulation of the background track being played. This made things like pausing the audio mid game execution possible.
* **Controls**
  + Using enumeration in the controls created a class that was easily portable between designs, as well as highly flexible. Switching from TankWars to Arkanoid was as simple as removing the “UP” and “DOWN” controls from the enumeration.
* **Debug State**
  + Allowed developers to toggle debugging statements that made development easier. By calling an instance of the debugState class, a developer could turn on or off built in debugging features.
* **Event Manager**
  + Manages observers from the gameEngine in order to notify objects and attach them to a keyListner.
* **InputHandler**
  + Manages keyListeners from the gameEngine for keyboard input.
* **Explosion / Explode**
  + Explosions are implemented in the same manner across games because both require explosion animations.
* **GameObject**
  + Allows the creation of a general gameObject with an id, previous x and y positions, BufferedImage, and a visible flag for drawing.
* **Physics**
  + A generic wrapper for collisions between gameObjects.
* **PowerUp**
  + Allows the gameEngine to create a list of powerUps in order to track which one the player has interacted with, and update the game accordingly.
* **Projectile**
  + Allows the gameEngine to create a list of projectiles and track them as update() is called. This allows each projectile to be individually tracked.
* **SoundManager**
  + Makes use of the AudioTrack class by creating a new instance of a soundFX when triggered by the gameEngine.