Project Summary

Call of 2D Zombies - a 2D Zombie Survival Game

Call of 2D Zombies is a two-dimensional zombie survival game with an overhead viewpoint. The player can move a character around the screen with keyboard input and aim a weapon with the mouse. Left clicking will fire a weapon in the direction of the mouse pointer. Zombies (squares) come in waves that increase in difficulty. The player has limited health, and each contact with a zombie decreases that health. The player’s goal is to shoot and kill the zombies, surviving for as long as possible.

This program will be built in Java. The framework for the game will come directly from: <http://www.gametutorial.net/article/Java-game-framework>

This framework includes four classes:

1. Window
   1. Creates a full screen window and places an instance of Framework on it
2. Canvas
   1. Canvas extends JPanel
   2. Listens for and tracks keyboard and mouse events
   3. Handles graphics and drawing methods
3. Framework
   1. Framework extends Canvas
   2. Is the main game loop that updates and draws the Game
   3. Has an instance of the Game class
   4. Handles a few mouse events
4. Game
   1. Has all game objects
   2. Tracks game statistics
   3. Handles updating game objects, components, and displays
   4. Handles collision detection
   5. Handles drawing game objects and display
   6. Handles all other gameplay

This game framework creates a window, sets up the drawing method, gets and tracks user input, and sets up the basic framework of the game loop. However, no game exists with the framework alone. I created the actual game features.

**What I Have Created:**

* Menu, cursor, and game map - images and displays
* A player that moves on an 2D plane based on user input (position, velocity, and acceleration)
  + Collision detection with enemies
    - Player enemy collisions decrease player health
    - If player dies, game is over
  + Collision detection with walls
    - Player bounces off the walls of the screen
* A weapon for the player that can shoot bullets in the direction of the mouse pointer (when mouse is clicked, if enough time has passed since the last click)
  + The weapon fire speed increases as rounds go on
  + Collision detection between enemies and bullets
    - Enemy bullet collisions subtract health from the enemy, possibly remove that enemy (if enemy dies), and remove the bullet
  + Collision detection between bullets and walls
    - Bullets that go out of screen are removed
* Enemy waves
  + Enemies come in waves, increasing in difficulty
    - More enemies spawn per wave as wave number increases
    - Enemy health increases and wave number increases
    - Player gets more ammo at the start of each wave
    - Enemies spawn from random positions outside of the screen
* Enemy AI
  + Enemies follow the player, some more delayed than others
  + Three types of enemies
    - Regular enemies – regular size, speed, and health
    - Smart enemies – smaller, faster, but have less health
      * In future versions, may follow predicted path of player
    - Large enemies – larger, slower, and more delayed, but have more health
* Statistics
  + Wave, kills, ammo, score, health
  + In game, pressing F1 toggles display of:
    - Screen display size
    - Current player position and velocity
    - Number of spawned enemies and number of enemies to be spawned

Algorithm

**Run Process (from start)**

* Program is started
* Main is called in Window, creates a window
* Window creates a Framework (extends Canvas) and adds it to its content pane
* Constructor of Framework calls the constructor of Canvas, sets the state to VISUALIZING, creates a thread for the game loop, and calls the GameLoop() of Framework
* GameLoop() on state VISUALIZING sets the frame height and width, sets the state to STARTING, and calls repaint()
  + Repaint() calls Draw(), which doesn’t draw anything in this state (VISUALIZING)
* GameLoop() on state STARTING calls Initialize() and LoadContent(), then sets the state to MAIN\_MENU, and calls repaint()
  + Initialize sets the font for the menu
  + Load content loads images, sound, etc. for the menu
  + Repaint() (Draw()) in this state (STARTING) doesn't draw anything
* GameLoop() on state MAIN\_MENU calls repaint()
  + If ESC is pressed, the game is exited
  + If left mouse is clicked, newGame() is called
  + Repaint() (Draw()) in this state (MAIN\_MENU) draws the menu text and images
* NewGame() creates a new Game()
  + Game() constructor sets the game state to GAME\_CONTENT\_LOADING
  + A new thread is created for the initialization of the Game class
    - In this thread, Initialize() and LoadContent() of the Game class are called, and afterward the game mode is set to PLAYING
      * Initialize sets the font, variables, array lists, etc. for the game
      * LoadContent loads images, sound, etc. for the game
  + While the game is being initialized on its own thread, the game loop of Framework (on GAME\_CONTENT\_LOADING) draws a "Game is Loading" screen
* GameLoop() now on PLAYING runs the regular game loop
  + - For each loop (for more detail, see game loop descriptions)
      * Update game
      * Collision detection
      * Draw
* When Player dies, the state is set to GAMEOVER
  + GAMEOVER in GameLoop() does nothing
  + GAMEOVER in Draw()
    - We draw the game over screen with "Game Over", game statistics (game.DrawStatistics(g2d, gameTime)), and "Enter to restart or ESC to exit"
  + If Enter is pressed, restartGame() of framework is called
    - RestartGame() resets the game time, calls game.RestartGame(), and sets the state to PLAYING
  + ESC closes the window

**General Game Loop Description**

1. Game // Has methods for updating, collision detection, and drawing for all game objects
   1. Player // player can update and draw itself and call update and draw on weapon
      1. Weapon // weapon can update itself and call update and draw on bullets
         1. Bullets // bullets can update and draw itself
   2. Waves // waves can update itself and call update on enemies
      1. Enemies // enemies can update and draw itself

Game gets updated in each game loop, which branches out and updates everything, then braches out and does collision detection (which does some more updating of its own), then branches out and draws everything.

**Detailed Game Loop Description**

* UpdateGame(long, point): void
  + limitMousePosition
  + UpdatePlayer
    - Update position
      * Did player hit off wall
    - Update posHist
    - Update weapon
      * Update weapon level
      * Is player shooting
      * Update bullet positions
        + Bullets off screen are removed

Done either here or in collision detection

* + UpdateWaves(long): void
    - UpdateWaves(): void
      * Spawning, etc.
    - UpdateEnemies(): void
      * Updates each enemies position and calculates new direction for each (accesses player position history - may need to be passed player history)
* Collision Detection
  + Check collision between bullets and enemies
    - If there is a collision, remove bullet, update enemy health, update enemy alive status, and possibly remove enemy
  + Gets players position and checks it with position of each enemy
    - If player collides with enemy, update player health, update player alive status
  + Bullets off screen are removed
* Draw
  + Draw(Graphics2D, Point, long): void
    - Draw background, player, bullets, enemies
  + DrawStatistics(Graphics2D, long): void
  + DrawMouse(Graphics2D, Point): void

**Class Descriptions**

**Window**

* Window extends JFrame
* Main method is in Window
* Constructor
  + Sets title
  + Sets to undecorated and full screen
  + Sets default close operation to exit\_on\_close
  + Sets the content pane of the window to an instance of Framework
    - Framework extends Canvas, which extends JPanel
  + Sets window visibility to true

**Canvas**

* Canvas extends JPanel and implements KeyListener and MouseListener
  + For drawing and getting user input
* Variables (two Boolean arrays)
  + One for tracking the current state of the keyboard (keyBoardState[525])
    - If a key is pressed, its corresponding index position in the keyboard array is set to true
    - If a key is released, its corresponding index position in the keyboard is set to false
  + One for tracking the current state of the mouse (mouseState[3])
    - Same for mouse
* Constructor
  + Sets buffer method, focusable, and background color
  + Sets cursor to blank (custom cursor will be added later)
  + Adds mouse and key listeners
* Methods
  + Canvas has methods for returning the Boolean value at an index position of the mouse or keyboard array
    - Public static Boolean keyboardKeyState(int key)
    - Public static Boolean mouseButtonState(int button)
  + Canvas also has methods for drawing Graphics to the panel

**Framework**

* Framework extends Canvas
* Framework is essentially the game loop
  + Updates 60 times per second
  + Draws at 60 FPS
* Variables
  + frameWidth and frameHeight (in pixels of screen) calculated in game state visualizing
  + secInNanoSec – 1 second in nanoseconds (used for conversions)
  + millisecInNanosec – 1 millisecond in nanoseconds (used for conversions)
  + GAME\_FPS – how many times we update and draw per second
  + GAME\_UPDATE\_PERIOD – used to calculate sleep time so game loop meets FPS
  + Enumeration GameState - all of the possible game states
  + GameState gameState – current game state
  + gameTime – elapsed time since start of game
  + lastTime – used for calculating elapsed time
  + Game game – actual game, game objects, etc.
  + Fonts and an image for the menu
* Constructor
  + Calls constructor of parent class
  + Sets game state to VISUALIZING
  + Creates a separate thread for the game loop
  + Calls the game loop
    - Game loop in state VISUALIZING gets the frameWidth and frameHeight
    - When that is done, the state is set to STARTING, which calls Initialize() and LoadContent()
* Methods
  + Initialize - sets font types for menu
  + LoadContent – loads the menu background image
  + GameLoop (ran ~60 times per second)
    - Game states (enumeration)
      * PLAYING
        + Calculates game time
        + Calls game.UpdateGame(gameTime, mousePosition())
        + Updates lastTime to current System time (in nanos)
      * GAMEOVER
        + Does nothing (for updating)
      * MAIN\_MENU
        + Does nothing (for updating)
      * OPTIONS
        + Does nothing
      * GAME\_CONTENT\_LOADING
        + Does noting (for updating)
      * STARTING
        + Calls Initialize(), which sets font type for menu
        + Calls LoadContent(), which loads the menu background image
        + Sets the game state to STARTING
      * VISUALIZING
        + Gets screen resolution to set frameWidth and frameHeight
        + Sets state to STARTING
    - Calls Draw
  + Draw (Graphics2d g2d) (called from GameLoop ~60 times per second)
    - Game states (enumeration)
      * PLAYING
        + Calls game.Draw(g2d, mousePosition(), gameTime)
      * GAMEOVER
        + Calls game.DrawGameOver(g2d)

Which draws “GAME OVER” with game stats

* + - * MAIN\_MENU
        + Draws main menu background image, game title, and game instructions
      * OPTIONS
        + Does nothing
      * GAME\_CONTENT\_LOADING
        + Draws a white screen with “Game is Loading”
  + newGame
    - Sets the game time to 0, sets last time to current System time, and creates a new Game object (which restarts everything)
  + restartGame
    - Sets gameTime to 0, last time to current System time, calls game.RestartGame(), and sets the game mode to PLAYING
  + mousePosition
    - Returns mouse position, unless position is null (off screen), in which case it returns (0,0)
  + keyReleasedFramework(keyEvent e)
    - Game states (enumeration)
      * GAMEOVER
        + ESC closes the program
        + ENTER restarts the game
      * PLAYING
        + F1 toggles the game stats (position, velocity, etc.)
      * MAIN\_MENU
        + ESC closes the program
  + mouseClicked(mouseEvent e)
    - Game states (enumeration)
      * MAIN\_MENU
        + Mouse BUTTON1 creates a new game

**Game**

* Variables
  + Player player
    - player handles player health, kills, position history, acceleration from user input, updating player position, drawing player, updating and drawing weapons and bullets, etc.
  + Waves waves
    - handles generating new waves, tracking wave number, spawning enemies, updating enemies, drawing enemies, etc.
  + Score – earned from killing enemies and surviving waves
  + backgroundImg – the game map grid image
  + mouseCursorImg – crosshair image
  + Fonts – for on screen display
  + Static Boolean displayF1stats for whether F1 stats should be drawn
* Constructor
  + Creates a separate thread for Initializing() and LoadContent() to be called
    - Initialize()
      * Creates a new Player
      * Creates a new Waves
      * Sets the score to 0
      * Sets font types
      * Sets displayF1stats to false
    - LoadContent
      * Loads game map grid image
      * Loads crosshair image
      * Calculates center of mouse crosshair image
    - Sets the game state to PLAYING
* Methods
  + RestartGame
    - Creates a new player
    - Creates a new waves
    - Sets score to 0
  + UpdateGame(gameTime, mousePosition)
    - If play is alive
      * Update player
      * Update player weapon, passing gameTime and mousePosition
      * Update waves, passing gameTime
      * Update enemies, passing player position history
      * Run collision detection
    - If player is dead
      * Set game state to GAMEOVER
  + Collision Detection
    - For each enemy, check if any bullets collide (if rectangles intersect)
      * If bullet does collide, subtract health from enemy, getAliveStatus to possibly remove enemy, and remove bullet
    - For each enemy, check if enemy collides with player (if rectangles intersect)
      * If enemy collides with player, subtract health from player, getAliveStatus from player and possible set game state to GAMEOVER
  + Draw(Graphics2D g2d, mousePosition, gameTime)
    - If player is alive
      * Draw background image
      * Draw bullets
      * Draw enemies
      * Draw player
      * Call DrawGameStatistics(g2d, gameTime)
      * Call drawMouseCursor(g2d, mousePosition)
  + DrawGameStatistics(Graphics g2d, gameTime)
    - If displayF1stats == true, draw
      * Frame size, x and y velocity, x and y position, total enemies, and total enemies to be spawned
    - Draw
      * Wave number, total kills, ammo, score, and health
  + DrawMouseCursor(Graphics2D g2d, mousePosition)
    - Draw mouse cursor with center of cursor image at mouse point
  + DrawGameOver(Graphics2D g2d)
    - Draw (game objects are drawn but not updated – appear frozen)
      * Background image
      * Bullets
      * Enemies
      * Player
      * Game title and “GAME OVER”
      * Game stats – wave, kills, score

**Player**

* Variables
  + HealthAtStart – player has 100 health at the beginning of each game
  + Health – current health, starts at 100
  + Boolean alive – updated during collision detection if player collides with enemy
  + totalKills – updated during collision detection if enemy dies
  + x and y position, and x and y velocity – changing variables
  + x and y acceleration and deceleration – constant
  + player size – x any y length of player rectangle
  + weapon – player weapon, which fires, updates, and draws bullets
* Constructor
  + Set x and y position to the passed x and y
  + Set health to 100
  + Set kills to 0
  + Set player size to 50
  + Set acceleration and deceleration to .2
  + Initialize all other variables
* Update
  + Call updateVelocity()
  + Call updatePosition()
  + Call updatePositionHistory()
* updateVelocity
  + IMPORTANT
    - dx positive moves to the right
    - dx negative moves to the left
    - dy positive moves down
    - dy negative moves up
  + Key input acceleration and deceleration
    - If D key is pressed, add ddxMove to dx (accelerate to the right)
    - If A key is pressed, subtract ddxMove from dx (accelerate to the left)
      * If neither A nor D is pressed, decelerate player dx
    - If magnitude of dx is greater than 6, set to 6 (max velocity)
    - ^ same for DY with W and S keys
  + If bullets hits off of any walls, inverse its velocity for whatever wall it hit
* updatePosition
  + Add dx to x and dy to y
* updatePositionHistory
  + Add current position to end of position history linked list
  + If linked list is greater than 60, remove that first (oldest) item
* UpdateWeapon
  + If mouse is pressed, call weapon.tryShoot, which shoots the weapon if enough time has passed since the last fired bullet
  + Call weapon.updateBullets, which moves each bullet
  + Call weapon.clampBullets, which removes any bullets that go off screen
* tryDamagePlayer
  + If enough time has passed since player was last damaged, damage again
* getAliveStatus
  + Update Boolean alive based on player health and return status
* getPositionHist
  + returns linked list of player history
* removeBullets
  + Gets linked list of Booleans from game, and sends them to weapon for which bullets should be removed
* drawBullets
  + Draws each bullet
* drawPlayer
  + Draws the player
* getBounds
  + returns player rectangle

**Position History**

* Variables
  + x and y for position
  + dx and dy for velocity
* Constructor
  + Sets x, y, dx, dy (player position when set)
* Methods
  + Getters and setters

**Weapon**

* Variables
  + ammo – ammo count
  + time of last created bullet and time between bullets – for handling fire rate
  + array list of bullets – current bullets being shot
* Constructor
  + Sets ammo to 10
  + Puts a half second delay between bullet firing
  + Initializes the array of bullets
* Methods
  + tryShoot
    - fires a bullet if enough time has passed
  + update bullets
    - calls update on each bullet in the array list
  + clamp bullets
    - removes any bullets that go out of the screen
  + remove bullets
    - gets passed an array list of bullets to be removed (from collision detection), and removes those bullets
  + update weapon level
    - called at the start of each new wave
    - increases fire rate
  + drawBullets
    - draws each bullet
  + getBulletRectangles
    - returns an array list of bullet rectangles for collision detection in Game
  + getBulletCount
    - returns total number of bullets (not ammo count)

**Bullet**

* Variables
  + Position x and y, velocity dx and dy
  + Speed and damage
  + size
* Constructor
  + Sets x and y (which, when created, is the players position)
  + Sets dx and dy, which are later changed
  + Sets the bullet size
  + Calls setVelocity and passed the mouse position
* Methods
  + updateBullets
    - adds dx to x and dy to y
  + draw
    - draws the bullet
  + setVelocity
    - based on player position when bullet is fired and mouse position when bullet is fired, a the dx and dy are calculated to create a velocity vector that will follow the mouse – i.e. the bullet x and y velocity is set to fire the mouse position
  + getters

**Waves**

* Variables
  + Wave number, for tracking wave number
  + Amount of each enemy to be spawned – calculated at beginning of each round
  + Enemy health levels – calculated at beginning of each round, increase each round
  + Ammo to add – calculated at beginning of each round, added to weapon ammo count
  + Array lists of enemies
  + Time of last created enemy (for each enemy type) – used for spawning
  + Time between enemy spawns (for each enemy type) – used for spawning
* Constructor
  + Initializes variables
  + Initializes enemy array lists
  + Sets state to CALCULATING NEW WAVE
* Methods
  + UpdateWaves(gameTime)
    - Game States
      * CALCULATING\_NEW\_WAVE
        + Increments wave number
        + Calculates enemies to be spawned (for each enemy type)
        + Increases enemy health (for each enemy type)
        + Calculates and adds ammo to weapon
        + Updates weapon fire rate based on wave number
        + Sets state to DELAY
      * DELAY
        + Waits 8 seconds to start spawning enemies
        + Sets state to spawning
      * SPAWNING
        + Tries to spawn each type of enemy

Enemies spawn if enough time has passed

* + - * + Calculates total enemies to be spawned

If there are no enemies to be spawned, state is set to FINISHING

* + - * FINISHING
        + Calculates total enemies left

When there are no enemies left

Add to game score

Set state to CALCULATING\_NEW\_WAVE

* + tryEnemySpawn – for each type of enemy
    - If enough time has passed to spawn an enemy, and enemy is spawned at a random location right outside of the screen
  + updateEnemies
    - calls calculate new direction (follow player) on all enemies
    - calls update on all enemies
  + damageEnemies
    - Collision detection done in Game returns a Boolean linked list of what enemies to damage in the array of enemies (for each enemy type)
      * This enemies are then damaged (health taken away)
      * We call get alive status, and if the enemy health is < 0
        + Alive status is set to false
        + Game score is increased
        + Total kills gets +1
  + DrawEnemies
    - Draws all enemies (of each type)
  + getEnemyRectangles
    - returns an array list of rectangles of all enemies for collision detection to be done in Game

**Enemy**

* Variables
  + Position x and y
  + Velocity dx an dy
  + Speed (for calculating dx and dy)
  + size
* Initializer
  + Sets health to passed value, passed from waves when the enemy is spawned
    - Enemy health increases as rounds go on
  + x set to passed x, y set to passed y
  + dx set to passed dx, dy set to passed dy
* Methods
  + updatePosition
    - adds dx to x, and dy to y
  + getters for x and y
  + damage
    - subtracts the passed amount from health
  + getAliveStatus
    - updates enemy alive status (true == alive, false == dead) based on enemy health
    - returns that status
  + getBounds
    - returns enemy rectangle for collision detection in Game

**BasicEnemy, SmartEnemy, LargeEnemy**

* Variables
  + None
* Constructor
  + Calls Enemy constructor
  + Sets speed and size
* Methods
  + calculateNewDirection – called at 60 times per second, for each game loop
    - gets passed the position history of the player
    - calculates dx and dy for enemy, such that enemy follows player
  + draw
    - draws the enemy

Data Design

As seen from the class descriptions, this program is entirely object oriented. The game loop (Framework) has an instance of Game, which gets updated, checked for collisions, and drawn for each loop. The game holds all objects and all data of the game, which include:

* Player // player can update and draw itself and call update and draw on weapon
  + Weapon // weapon can update itself and call update and draw on bullets
    - Bullet // bullets can update and draw itself
* Waves // waves can update itself and call update on enemies
  + - Enemies // enemies can update and draw itself
* Array lists are used for just about everything
  + Enemies, bullets, player position history, etc.
* Integers are used to keep track of game statistics
* We create a Graphics2D g2d that we essentially pass around to the player, enemies, statistics, etc. to draw everything on it then put it on the screen (for each frame)
* Images are used for the menu, the in game background, and crosshair

Overall, the most common data type used is the java array list, and the overall data structure is object oriented.