Team: Alpha-Bravo

Members: Anthony Martinez (Team Leader), Dustin Chhum, Howard Cho, Luis Rodriguez

**List of Entity Methods**

1. **Game**
   1. **Method**: connectToDB

**Description:** Establishes a connection to the game database to perform CRUD operations.

**Pre-Conditions:** Database must be able to receive request.

**Post-Conditions:** A connection is established to the database

**Signature:** String connectToDB(String userName, String password);

* 1. **Method:** saveGame

**Description:** Saves game data to the game database and returns a boolean to indicate result.

**Pre-Conditions:** Database connection must be established.

**Post-Conditions:** Game data is saved to the user’s entry.

**Signature:** boolean saveGame(GameObject savedObject);

* 1. **Method:** loadGame

**Description:** Loads the game from the gameDB (database).

**Pre-Conditions:** Database connection must be established.

**Post-Conditions:** Game is returned.

**Signature:** Game loadGame(String query);

1. **Game Database**
   1. **Method**: selectData

**Description**: Retrieves data from the game database.

**Pre-Conditions:** A connection to the game database must be established.

**Post-Conditions:** A game object is returned from the database.

**Signature:** GameObject selectData(String query);

* 1. **Method**: updateData

Description: Updates data already present in the game database.

**Pre-Conditions:** Database connection must be established and the entry being updated must exist.

**Post-Conditions:** The game database is updated.

**Signature:** boolean updateData(String query);

* 1. **Method**: deleteData

**Description**: Deletes data contained in the game database and returns a boolean to indicate its success.

**Pre-Conditions:** Database connection must be established and the entry being deleted must exist.

**Post-Conditions:** Database entry is deleted.

**Signature:** boolean deleteData(String query);

* 1. **Method**: createData

**Description**: Creates a new entry in the game database.

**Pre-Conditions:** Database connection must be established.

**Post-Conditions:** A new record is entered in the database.

**Signature:**

1. **Main Character**
   1. **Method:** playerWalking

**Description:** Lets the main character move either left or right.

**Pre-Conditions:** Character has to have loaded.

**Post-Conditions:** Character moves either left or right.

**Signature:** void playerWalking();

* 1. **Method:** playerJump

**Description:** Lets the main character jump

**Pre-Conditions:** Character has to have loaded.

**Post-Conditions:** Character jumps up in the air.

**Signature:** void playerJump();

* 1. **Method:** playerCrouch

**Description:** Lets the main character crouch down.

**Pre-Conditions:** Character has to have loaded.

**Post-Conditions:** Character crouches down.

**Signature:** void playerCrouch();

* 1. **Method:** playerAim

**Description:** Lets the main character aim its weapon.

**Pre-Conditions:**Character has to have loaded and weapon has to have loaded.

**Post-Conditions:** Character aims its weapon.

**Signature:** void playerAim(double x, double y);

* 1. **Method:** playerShoot

**Description:** Lets the main character shoot its weapon.

**Pre-Conditions:** Character has to have loaded and weapon has to have loaded.

**Post-Conditions:** Character shoots its weapon.

**Signature:** void playerShoot();

* 1. **Method:** playerMelee

**Description:** Lets the main character melee.

**Pre-Conditions:** Character has to have loaded.

**Post-Conditions:** Character melees.

**Signature:** void playerMelee();

* 1. **Method:** playerDie

**Description:** Handles the player’s death.

**Pre-Conditions:** Character has to have alive.

**Post-Conditions:** Character dies.

**Signature:** void playerDie();

1. **Enemy**
   1. **Method:** enemyAI

**Description:** Determines the actions of the enemy.

**Pre-Conditions:** Enemy must have spawned.

**Post-Conditions:** Enemy does what the AI determines.

**Signature:** void enemyAI();

* 1. **Method:** enemyAttack

**Description:** Lets the enemy attack the player.

**Pre-Conditions:** Enemy must have player in sight.

**Post-Conditions:** Enemy attacks player.

**Signature:** void enemyAttack();

* 1. **Method:** enemyJump

**Description:** Lets the enemy jump.

**Pre-Conditions:** Enemy must have spawned.

**Post-Conditions:** Enemy jumps.

**Signature:** void enemyJump();

1. **Inventory**
   1. **Method**: activeWeapon

**Description**: Selects the current weapon that is attached to the character entity. This will determine the capabilities of the weapon.

**PreConditions**: Inventory must have available weapons to lead.

**PostConditions**: The weapon will be the current weapon.

**Signature**: void activeWeapon(int weaponPosition);

* 1. **Method**: insertWeapon

**Description**: Inserts a weapon into the inventory.

**PreConditions**: Inventory must have not be full.

**PostConditions**: Inventory will be incremented.

**Signature**: void insertWeapon(Weapon weaponObject);

* 1. **Method**: dropWeapon

**Description**: Deletes the current weapon from inventory.

**PreConditions**: The character must currently have an active weapon.

**PostConditions**: The active weapon is deleted from inventory.

**Signature**: void dropWeapon();

* 1. **Method**: isFull

**Description**: Check if inventory list if full and returns a boolean.

**PreConditions**: The main character must be loaded.

**PostConditions**: A boolean will be returned.

**Signature**: boolean isFull();

1. **Weapon**
   1. **Method**: fireWeapon

**Description**: This will shoot a projectile .

**PreConditions**: Ammunition must not be empty.

**PostConditions**: Ammunition will be decremented.

**Signature**: void fireWeapon();

* 1. Method: setAmmunition

Description: Adds an amount of ammunition for a weapon.

PreConditions: Inventory must have at least one weapon in it.

PostConditions: Ammunition will increase.

Signature: void setAmmunition(int amountOfAmmo);

* 1. **Method**: isEmpty

**Description**: Checks if ammunition is empty. If weapon is empty, then the weapon will disappear.

**PreConditions**: Character must have an active weapon.

**PostConditions**: A boolean is returned.

**Signature**: boolean isEmpty();

1. **Health**
   1. **Method:** getHealth

**Description:** Returns the health.

**Pre-Conditions:** Character/enemy must have loaded.

**Post-Conditions:** Returns health.

**Signature:** void getHealth();

* 1. **Method:** setHealth

**Description:** Sets the health to max.

**Pre-Conditions:** Character/enemy must have loaded.

**Post-Conditions:** Health is at max.

**Signature:** void setHealth(int maxHealth);

* 1. **Method:** healthGain

**Description:** You gain health.

**Pre-Conditions:** Character must have loaded and some health has to have been lost.

**Post-Conditions:** Player gains health.

**Signature:** void playerHealthGain(int amtGained);

* 1. **Method:** healthLost

**Description:** You lose health.

**Pre-Conditions:** Character/enemy must have loaded.

**Post-Conditions:** Player/enemy loses health.

**Signature:** void healthLost(int amtLost);

1. Obstacle
   1. **Method:** setObstacle

**Description:** Sets an obtacles type, dimensions, and map position.

**Pre-condition:** Dimensions and position must be valid.

**Post-Condition:** Dimension and Position for an obstacle will set.

**Signature:** void setObstacle(string type, int posX, int posY, int height, int width)

* 1. **Method:** getObstacle

**Description:** Returns an obstacle.

**Pre-Condition:** Valid obstacle must have been created.

**Post-Condition:** Returns an obstacle.

**Signature:** Obstacle getObstacle();

* 1. **Method:** placeObstacle

**Description:** Places a obstacle on the level to impede the player from proceeding.

**Pre-Condition:** A valid game and map must have been generated.

**Post-Condition:** An obstacle will be placed on the level.

**Signature:** void placeObstacle(Obstacle obstacle);

* 1. **Method:** playerObstacleContact

**Description:** Determines if player made contact with obstacle.

Will be used to determine if damage will be dealt to player from obstacle.

**Pre-Condition:** Player/Obstacle must have been loaded.

**Post-Condition:** Return if player made contact with obstacle.

**Signature:** bool playerObstacleContact(Player player, Obstacle obstacle);

* 1. **Method:** setObstacleDamage

**Description:** Places an amount of health certain obstacles can deal to the player.

**Pre-Condition:** A valid obstacle must have been created.

**Post-Condition:** The obstacle now has damage to deal.

**Signature:** void setObstacleDamage(int damage);

* 1. **Method:** getObstacleDamage

**Description:** Returns the obstacle damage number.

**Pre-Condition:** A valid obstacle must have been created.

**Post-Condition:** Return the obstacle damage number.

**Signature:** int getObstacleDamage();

1. **Platform**
   1. **Method:** setPlatform

**Description:** Platforms dimension and position will be set.

**Pre-Condition:** Dimensions and position must be valid.

**Post-Condition:** A Platform’s position and dimensions are set.

**Signature:** void setPlatform(int posX, int posY, int height, int width);

* 1. **Method:** getPlatform

**Description:** Returns the platform.

**Pre-Condition:** Must be a current valid obstacle.

**Post-Condition:** Returns a platform.

**Signature:** Platform getPlatform();

* 1. **Method:** placePlatform

**Description:** Places a platform on the level.

**Pre-Condition:** A valid game and map must have been generated.

**Post-Condition:** A platform will be placed on the level.

**Signature:** void placePlatform(Platform platform);

1. **Level**
   1. **Method:** setLevel

**Description:** The current level is set.

**Pre-Condition:** Current level must have been created.

**Post-Condition:** The current level/map the player is in is set.

**Signature:** void setLevel(Level level);

* 1. **Method:** getLevel

**Description:** Returns the current level

**Pre-Condition:** A valid level must have been set.

**Post-Condition:** Returns the current level.

**Signature:** Level getLevel();

* 1. **Method:** generateLevel

**Description:** The level generates with all other components needed like the

player, enemies, platforms, and items.

**Pre-Condition:** Level must be valid.

**Post-Condition:** Level is generated onto screen.

**Signature:** void generateLevel(Player player,Enemies enemy, Weapons weapon,

Items item, Obstacles obstacle, Platforms platform);

* 1. **Method:** setLevelEnding

**Description:** Places a marker at the end of the level.

**Pre-Condition:** Level must have been generated.

**Post-Condition:** Places a marker signaling this is the end of the level.

**Signature:** void setLevelEnding(int endPosX, int endPosY)

* 1. **Method:** levelEndingReached

**Description:** Determines if player has reached the end of the level.

**Pre-Condition:** The end level marker must have been created.

**Post-Condition:** Returns a true or false value if player reached the end.

**Signature:** bool levelEndingReached(int playerPositionX, int endPositionX);

* 1. **Method:** startTimer

**Description:** A timer will start at the start of the game.

**Pre-Condition:** A successful level was loaded.

**Post-Condition:** A timer starts to keep track of the time.

**Signature:** void startTimer();

* 1. **Method:** stopTimer

**Description:** The current level timer is stopped once player completes level.

**Pre-Condition:** Timer must have been started.

**Post-Condition:** Will give the final time the player took to complete the level.

**Signature:** void stopTimer();

* 1. **Method:** getTime

**Description:** Returns the time from the timer created.

**Pre-Condition:** A timer must have successfully started.

**Post-Condition:** Returns Time once player died or completed the level.

**Signature:** long getTime();

* 1. **Method:** pauseTimer

**Description:** When player pauses game the timer will pause.

**Pre-Condition:** There must have been a valid timer and player must have paused.

**Post-Condition:** Pauses the timer during player pause.

**Signature:** void pauseTimer();

* 1. **Method:** resumeTimer

**Description:** When player unpauses the game the timer will resume.

**Pre-Condition:** Player must have been paused.

**Post-Condition:** Resumes the timer after player unpauses to continue game.

**Signature:** void resumeTimer();

* 1. **Method:** giveRank

**Description:** A rank will be given once player reaches the end of level.

**Pre-Condition:** Player must have reached end of level.

**Post-Condition:** Rank is given to player.

**Signature:** int giveRank();