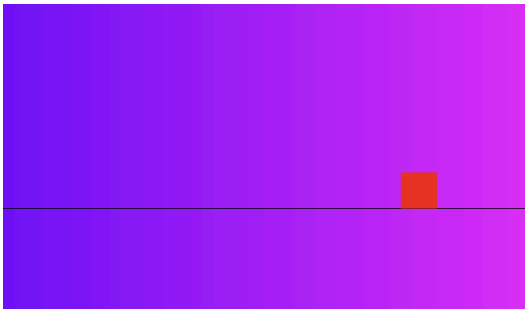
Geometry Dash Read Me and Revision History

Caitlin O'Leyar, Aadiva Rajbhandary, and Rizouana Prome

Project Description

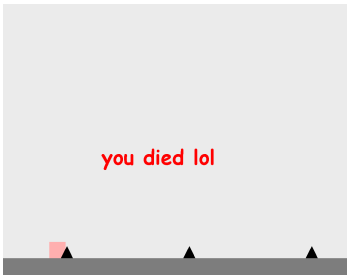
We have made our own version of the game Geometry Dash. In it, the player tries to complete a level by jumping over and on various obstacles. These obstacles include blocks of varying heights, double-jump stars, spikes, and platforms. There is a start menu where the player can select which sprite they wish to play as. The player can press a pause/resume button on the top left of the game at any time. When the player collides with an obstacle, they die. The goal is to reach the end without colliding. The player is given unlimited attempts to complete the level.

Revision History



Our initial step for creating a geometry dash game was making a map where a square would be moving in a straight line and could jump. Applying gravity to the square was a bit tricky. At first, when we pressed the spacebar, the square would just move up to a higher position with no smooth motion. We tried looking at many tutorials and guides (on YouTube, StackExchange, etc) to figure out how to implement gravity, but none of them worked. So, eventually, we asked ChatGPT how to implement gravity. While its code did not fully work, its ideas helped us come up with the GRAVITY and JUMP_SPEED ints as well as the isJumping boolean.

Next, we implemented a rectangular obstacle that required the square to jump over it. At first, if the square bumped into the obstacle, the player would simply stop moving forward. But we changed it so that if the square bumped into the obstacle the game would end and the player would die.



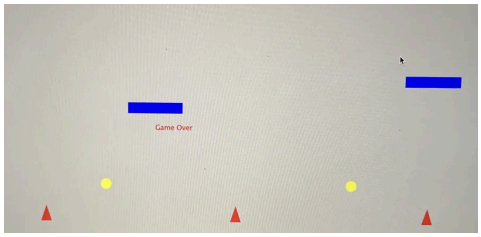
Eventually, when we were adding in more obstacles, we realized that we needed a way to move the camera view. At first, we tried having a long stationary map that the player could move along, but we had a lot of issues doing that. So, we made an integer called cameraX. While the level is playing, the cameraX is set to the playerSpeed. Each object in the map has its position updated by cameraX, so the objects gradually approach the player square.

We also implemented a play and pause button to allow the player to take breaks if needed. When the player presses the pause button, each aspect of the game (applying gravity,

moving objects, checking collision, background music, etc) stops until the player resumes the game.

Then, we tried to implement a star that would allow the player to double-jump over taller obstacles. Originally, it was just a circle, but we wanted to make it star-shaped. We ran into a lot of issues trying to draw the stars and making the array point move in and out of the center. To fix the issue of drawing the star we ended up using the modulus function. We set the condition to draw the outside if it was even and draw the inside if it was odd. At first, when we tried to make the player jump when colliding with the star, we just set isJumping to false, but then the player would start hovering mid-air at the star's elevation. So, to ensure the double jump, we used the isOnStar boolean to create a new state where the player could jump again without treating it like the ground level.

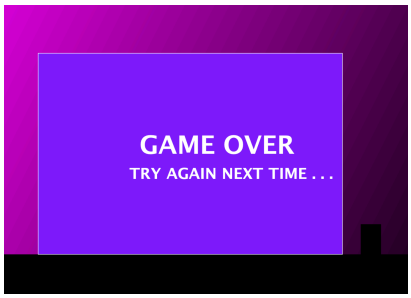
Next, we implemented spikes as new obstacles, which was fairly easy to do. At first, updating the position of the spikes was a bit tricky because, with the regular blocks, we could just do the xPosition - cameraX. But the spikes and the stars were created using arrays. So, we had to update each array position separately. We did this by doing xPosition - cameraX for each point of the array.



When we tried to implement platforms to help the player get through a combination of spikes, we had a lot of issues with the collision of the platform as the player would not land on the platform and fall right through it. To fix this issue we created conditions that allowed the player to land on top of the platform and move.

At this point, we started adding images and sounds to make the game more aesthetically pleasing. This includes adding a background image, a level song, sound effects (colliding with an object, completing the level, jumping on a star, opening the start menu), and more.

At first, we just used regular fonts and Strings for the text messages ("You Won," "Game Over," "Number of Attempts," etc), but we kept having formatting issues. For example, Caitlin picked out a specific font that was available on Eclipse, but when Rizouana tried to run it on Visual Studio Code, all the formatting was wonky. So, to get around this issue, we used a font generator to just make all of the text messages images.



Next, we created a start menu that only shows up when the player first launches the game. We also made it so that players could select which sprite they wished to play as, which was easy.

For the simple sound effects like the collision and win noises, just using the clipName.start(); worked fine, but it was not working as well for the background music. The intention was for the music to play when the level began, pause whenever the player pressed pause and resume when they pressed play, and end when the player won or lost. Looking into it, it seemed like making a thread would help with this. The general structure of the thread used in the code (new Thread(() -> { }) was guided by ChatGPT, but the try-catch functionality and the other aspects of the startMusic(); method were not.

There were a lot of issues with adding and layering music effects as well. When we first implemented a sound to play when the start menu popped up, the background music of the level would not play until the second attempt. To fix this, in the action performed by the start button, we put that the menu noise would stop and the music would begin. We also added an if statement to ensure that the music would be played in attempts other than the first one.

Once all of that was in place, we started organizing the map by combining the different obstacles into a new method called generateMap(). To create the spacing and obstacles that we wanted, we had to figure out the spacing of each obstacle and manually add each new object to the obstacle arrays.

When we finished with everything else, we implemented a simple label that displayed the player's progress through the level.

Game Design

Our main class is called GeoDash, where most of our code is. All the code is in one file, so the rest of the classes are private. These are called Obstacle, TallObstacles, Star, Spikes, Platform, and Win. All of these are pretty simple classes that act as objects in the game's level. The Win class represents the line that signifies that the player has completed the level.

Our program's main method has only two lines of code. The first line is to create a new instance of the GeoDash class called square.

The GeoDash constructor method calls the init() method. Here, the initial values of instance variables are set. This includes the playerSpeed, velocityY, and boolean variables like isJumping, isOnStar, isOnPlatform, gameOver, gameWon, isMenu, isPause, and isRunning. Out of those, only isRunning is initially set to true. If times is equal to 0 (so, before the restart() method has been called), then isMenu and isPaused are true, so the start menu displays. Otherwise, those booleans are false. A timer is initialized. The integers playerX, playerY, and cameraX (which set positions) are given values. Array lists of each of the different objects are created (obstacles, stars, spikes, tallObstacles, and platforms), the method generateMap() is called (which sets the positions of all of the objects), and the win line is created. Finally, the methods images(), audioStreams(), and font() are called, which initialize the assets.

Back in the main method, the second line calls square.restart();. The restart() method is where the game's JFrame is found. Here, there is the pause button, character selection buttons, start button, restart button, and number of attempts label.

In the restart() method, the first two lines of code are attempts++ and times++. The integer attempts keeps track of how many times the player has died before reaching the end. If the player reaches the end of the level and restarts the level, the number of attempts is reset. The times integer keeps track of how many total times the player has died/attempted the level. If the player reaches the end and restarts the level, the times integer is simply updated and not reset. The start menu will only appear when the number of times is equal to 1 (so when the game is first launched). This menu consists of a menu background image, the character selection buttons, images with text giving instructions, and a start button. If the player clicks one of the character selection buttons, that sprite will be the sprite that the player plays as throughout their level attempts.

Once the start button is pressed, the menu disappears and the actual game begins. Here, the game music will begin because it plays while isRunning is true and isPaused is false. If the player presses the pause button, the pause() method is called. Most of this method involves visual changes (such as changing the button image and changing the text from "Playing" to "Paused"), but it also switches the truth value of the isPaused boolean.

Most of the code is called within the actionPerformed(Action Event e) method. Here, actions continue while the isRunning is set to true, which only changes to false if the player dies from a collision. If the game is not paused, movePlayer(), applyGravity(), moveCamera(), updatePercentageLabel(), and methods to check collisions are performed. The only method that plays if isPaused is false is repaint().

The method movePlayer() simply changes playerX to be += to playerSpeed. This is to move against the movement of cameraX, called from the moveCamera() method, which moves objects in the

direction of the player. Combined, these create the effect that the player square stays in place while all of the objects approach the player.

The method applyGravity() is what makes the player appear to have gravity. If the player is jumping, the velocityY integer is updated to be -= to the integer GRAVITY, and the playerY integer is set to be -= to velocityY. This gives the appearance that the player is gradually falling back down after they have jumped. If the player reaches the level of the floor, then the playerY position is set to be on top of the ground, velocityY is updated to be 0, and isJumping is set to be false because the player is no longer in the air.

The isJumping boolean is updated from within the keyPressed(KeyEvent e) method. Here, if the player presses the spacebar, and they had not already been jumping, the boolean is updated to be true and the velocityY is set to JUMP_SPEED. However, being on the ground is not the only time that the player can jump. They can also jump if they are colliding with a star or on top of a flat surface (obstacle, tall obstacle, or platform). So, if isOnStar or isOnPlatform is true, then the same actions are performed as if they were on the ground (isJumping is set to true and velocityY is set to JUMP_SPEED).

The updatePercentageLabel() method simply changes the value of the label in the upper right-hand corner of the screen that displays the player's percentage through the level. It starts at 0% and goes to 99% (before disappearing when the user crosses the win line). The percentage through the game is calculated by (playerX-300)/(win.x-300)*100. So, the player's position (subtracted by 300 because the player originally begins at 300) is divided by win.x (subtracted for the same reason. And the multiplier of 100 is to make a decimal value into a percentage point.

The collision methods check collisions for each of the different types of obstacles and respond accordingly. If the player collides with a platform or is on top of a safe flat surface, their playerY position is set to be on top of that obstacle, their velocityY is set to 0, and isOnPlatform is set to true. If they collide with a star, isOnStar is set to true. If they collide with an unsafe surface like the side of a block or a spike, gameOver is set to true, the deathNoise plays, and the method endOfGame() is called. If they collide with the win line, gameWon is set to true, the winNoise plays, and the method endOfGameWon() is called.

The endOfGame() method makes the restart button visible and makes isRunning false, which stops all other activity in the code. The endOfGameWon() method makes the restart button and number of attempts visible, resets the attempts, and makes isRunning false.

Most of the visuals are created in the paintComponent(Graphics g) method. This draws the background image, the player, the floor, and all of the obstacles. Depending on the conditions, it will paint other things too. For instance, if the menu is open (so times == 1 and isMenu == true), then the

menu background, game name, and instructions are painted. A box to show which sprite has been selected is also displayed. The paint component also has if statements to draw the paused and playing images, the game over screen, and the game win screen.

Finally, if the restart button is pressed, then the current frame is disposed, the images and audio are flushed from the stream, a new instance of GeoDash is created, square2, and square2.restart() is called, which allows the player to play again and again.

Flourishes

Our base game consisted of the gravity, default block obstacles, keyboard/mouse interaction (space bar for jump, mouse clicks for the replay, pause, play, start), the winning/losing mechanics (lose when you collide with an obstacle, win when you reach the end), gravity, and collision detection (with each of the obstacles in a different way).

Each of the group members implemented a couple of flourishes. Aadi created the tall obstacles and stars. Rizouana created the spikes and platforms. Caitlin created the start menu with the character selection, the replay function, and added the sound effects. Making the game look cool overall through images was a group effort.

Citations

Menu image (darkened-menu-background): gamedevmarket.net/asset/lake-background

Level image (darkened-background): vecteezy.com/vector-art/6316482-alien-planet-game-background

Generated text images (gamename, GameOver, instructions, instructions2, NumberOfAttempts, Paused, Play_Again, Playing, start, YouWin): gdcolon.com/gdfont

Player sprites (squareIcon, squareIcon2, squareIcon3, squareIcon4, squareIcon5, squareIcon6): gdbrowser.com/iconkit/

Font used for the attempts integer (Bubblegum.ttf): dafont.com/

Sound effect (game_start): pixabay.com/sound-effects/search/game/?pagi=2

Sound effects (death_sound, level_complete_sound): myinstants.com/en/search/?name=geometry+dash

Level background music (background_music): downloads.khinsider.com/game-soundtracks/album/geometry-dash/1-27.%2520Back%2520On%2520Track%2520%2528GD%2520Cut%2529.mp3

Some of the buttons (replay, play_button, pause_button) were made by Aadi