### Easy

* Name your game.
* Change the dimensions of the game window.
* Find your own custom artwork for blocks.
* Find your own custom artwork for the hero.
* Find your own custom artwork for enemies.
* Create your own custom artwork for one unanimated entity in the game. Blocks and entities that only flip when they change directions are not considered animated. <http://www.piskelapp.com/> is a good site for this.
* Create your own custom artwork for at least three unanimated entities in the game.
* Find your own fonts.
* Find your own sound effects.
* Find your own theme music for a level.
* Use different theme music for each level you make.
* Customize the background layer for a level.
* Customize the scenery layer for a level.
* Create a custom splash screen that does more than just show the name of the game.
* Improve the display\_message function so that the text is in a box making it easier to read against the background.
* Change point value for coins.
* Make power-ups give points to the player.
* Make a bad item which takes points away from the player.
* Add a prize other than a coin and give it a point value.
* Display hearts/max rather than just hearts.
* Update the display\_stats function so that it shows the name of the current level.
* Add a victory sound that plays when the game is won. This should play instead of the normal end-of-level sound.
* If you kill enemies (in the 'Hard' section), award different point values based on the type of enemy killed.

### Medium

* Create your own custom artwork for one animated entity in the game. Entities that only flip when they change directions are not considered animated. Animation should include multiple frames such as when the hero runs. <http://www.piskelapp.com/> is a good site for this.
* Create your own custom artwork for at least three animated entities in the game.
* Design a complex standard level. Your level should be at least 60 blocks long.
* Create levels which takes place on a different planets. You can modify the gravity and terminal velocity in the level data to create the physics you want. (A gravity of less than 1 will make the world 'bouncier'.) Use space-themed backdrops for your levels.
* Create a game with at least 4 levels total. Levels should be significantly different in layout and each should be at least 60 block long.
* Display lives in the format [character icon] x [number of lives].
* Display actual hearts to show health. Show empty hearts when health is not full.
* Let the 'S' key toggle sound effects (jump, get coin, etc.) on and off. Show a little speaker/mute icon with game stats to indicate the current state of sound effects.
* Let the 'M' key pause and unpause level theme music. Show a little note icon (crossed out or uncrossed) with game stats to indicate the current state of music.
* Create invincibility (star) power-up. (The optional arguments on the play\_sound function can help you match an invincibility sound length to the invincibility period.)
* Invent your own power-up which lasts a limited amount of time or until the player hits an enemy or completes a level.
* Add a falling image to your character animation.
* Increase the max hearts a player can have when some milestone is achieved.
* Track coins separately from the score. (They can still be worth points.) Give an extra life when a number of coins is earned. Reset the coin count after a life is given.
* Invent a "power-up" which has a negative consequence on a player other than reducing hearts, lives, or points.
* Put gaps in the blocks that run along the bottom of the level. Then make a player die when they fall through the bottom of the world. You'll need to make sure enemies that fall through are also removed from the game. Pygame's sprite.kill() function will be useful for this.
* Make a credits screen for when the player wins the game.
* Add a PAUSE stage to the game which is activated when the player presses 'p' (or a button on the joystick). All movement and time should stop during a pause stage. Pressing 'p' again should resume. Be sure to show a message indicating the game is paused.
* Use a modifier key/button that makes you run faster (or perhaps jump higher) while the key/button is held.
* Add a third parallax layer that scrolls at a different speed than the other layers. Be sure that closer layers always scroll faster than far away layers.
* Create a custom enemy with unique behavior regarding movement.
* Create cover art for your game. Save it as a PNG file. Print it in color and I'll post it in the room.

### Hard

* Change the game so that it uses the XBox controller instead of the keyboard. Be sure to map all game functions such as start and pause to the controller and not just player control. (<https://github.com/joncoop/pygame-xbox360controller>)
* Create an underwater level. You can use low gravity and low terminal velocity to slow down the up and down motion. You should use the jump key to swim up and then let the player slowly drift down. The tricky part is getting the jump to work whether or not you are standing on a block. Be sure that jumping still functions normally in land levels.
* Create a game with at least 8 levels total. Levels should be significantly different in layout and each should be at least 60 block long.
* Kill enemies when you land on them. You'll need to check which direction you hit the enemy from in process enemies. When you do kill an enemy, make the hero bounce up a bit rather than just fall through the enemy. Also, be sure that the hero doesn't become invincible for a brief period after a kill as he does when normally colliding with an enemy. Award points for killing an enemy.
* If you kill enemies, create enemies that take more than one hit to kill. You should either bounce high enough off of the enemy to force a delay between first and second hits or give an enemy a brief invincibility period after a hit.
* Give points for getting the flag at the end of the level. Do so in a way that landing higher on the flagpole earns more points. One way to do this could be to check the distance between the hero and the ground when the flag is reached and devise a formula to award points. Another way is to assign a value to flag pieces as they load and give points for the flag piece intersected. Notice that the flag pieces load from highest to lowest in the level class.
* Make the game save high scores to a text file in a data folder. The high score should be displayed on the splash screen and on the end screen. Be sure that the end screen makes the player aware if they achieve a new high score.
* Show time on the stats layer. Give a time bonus for completing a level. Have the hero die if the level is not completed in a set amount of time.
* Animate a scenery element such as clouds or flickering torches. (This is probably best accomplished by creating another layer that scrolls along with the background or scenery layer rather than trying to blit directly on one of those layers.)
* Create a foreground layer. The foreground layer should scroll at the same speed as the active layer. However, the player should run behind items on the foreground layer (perhaps tufts of grass or trees). Rather than use an image for the background like the scenery and background layers, draw items using grid items in the same manner as blocks, coins, etc. Just be sure to treat it like the inactive layer. Don't update it each iteration fot he game loop. The foreground layer should be purely decorative like the scenery and background layers. It is not necessary to make the hero interact with any items in the foreground.
* Utilize vertical scrolling in a level. You'll need to modify the calculate\_offset function to get vertical scrolling to work.
* Create a Warp object in the game that transports the player instantly to another part of the level. Warps should be declared in the level.json file and indicate both the location of the warp item and the location that the player will warp to. (A four-element list would work well for this.) Make the warp either a necessary part of solving a level or a way for a player to get otherwise unreachable prizes. Don't try to make two-way warps. One-way will be much easier.

### Very hard

* Add ladders to the game. If a player is on a ladder, don't apply gravity. Assign vy by player input instead. Also disable jumping while on a ladder. You should use animated climbing images too.
* Put switches in the game that open and/or close areas of a level.

### Super hard

* Create a Chest object with a boolean attribute locked set to True when it is initialized. Then create a Key object that a player can acquire. If the player intersects a locked chest with a key, then locked should be set to False and the image should be updated. Spawn a prize that the player can acquire in the grid location directly above the chest when it is unlocked. Be sure to take away the players key when the character respawns after dying or at the start of a new level.
* Give your player a gun. Let your player kill enemies by shooting them. You'll need to create Bullet objects which spawn at the hero's gun and travel in the direction the hero is facing. Bullets shouldn't travel indefinitely. Limit the number of ticks a bullet will exist before it calls the kill() function on itself. Enemies can be given a process\_bullets function. (You might need to make the shooter/hero a bullet parameter so that when the bullet hits an enemy, points can be awarded.)
* Make a sign you can read or a character that 'speaks'. Display a message when the hero intersects the sign or speaking character while holding the up arrow (or up on the d-pad). Don't use the default display\_message function. Put a function in the Sign/SpeakingCharacter class that makes a popup that looks like text on a sign or perhaps a speech bubble. Be sure that an action by the player can dismiss the message.
* Make a special level at the end of the game with a boss that needs to be defeated. The boss should take multiple hits to kill, have a unique behavior compared to other enemies, and do something to try to kill the hero. (Shoot stuff, breathe fire, ...?)

## Grading

Use game.py file to create a platformer game. Points will be awarded for each feature as follows:

* Easy: 5 points each
* Medium: 10 points each
* Hard: 15 points each
* Very hard: 20 points each
* Super hard: 25 points each

This project will count as two major assessments. When you reach 100 points, you can show me your work to get credit for the first assessment. Then continue adding more features to earn points toward your second major assessment. There are over 600 points available, so you should have plenty of options for earning points, either by choosing to complete many easy features or fewer difficult features.