Mini Project Four - Block Builder

Kathryn Hite and Alix McCabe

March 2015

1 Overview

In this project, we sought to make an interactive leg-building environment using OpenCV's color tracking. We wanted to make a fun play-space for users that enabled them to move Legos with an object in the real world instead of their computer keys. We felt this added level of interaction would make the program more engaging for users, in addition to aligning with our learning goals for this project.

2 Results

We successfully accomplished our main goals of creating an interactive Lego-building environment based on user interaction with the computer camera. The user input is a blue object held in front of their computer's camera. This object becomes aligned with a virtual lego on the screen. When they begin to move their object, the Lego on the screen will move with it until the user has moved the lego to its desired position.

The point of the game is to build Lego creations in a virtual environment, and when they've moved their Lego to the desired location, users can press the space bar to release the Lego into the environment. Once they've dropped a Lego, another Lego is generated with a random color. From here, users can continue moving, dropping, and generating differently colored Legos until their Lego masterpieces are complete!

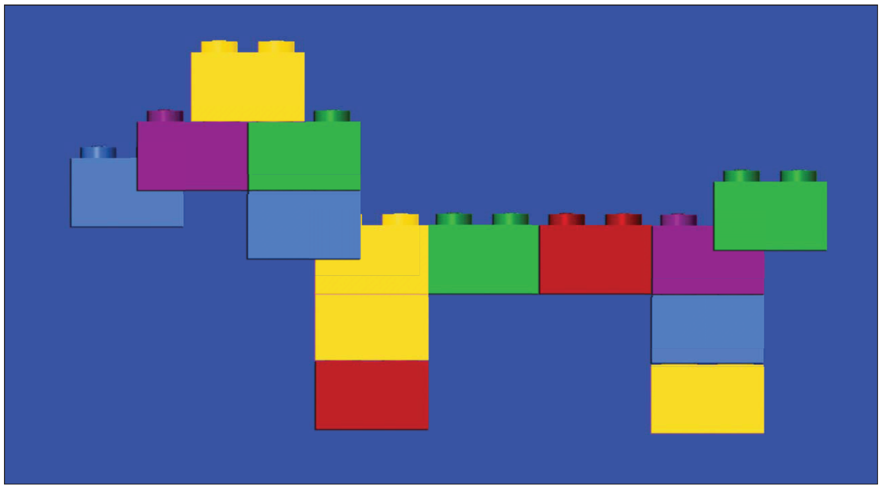

Figure 1

3 Implementation

Our main user input function is called "tracker()" and is what our code uses to update the x and y positions of the current Lego blocks. It uses OpenCV's color tracking software that finds the central x and y coordinates of a blue object as tracked by a computer camera.

We used classes to implement most of our code, including classes for the state of the game, the environment of the game, the components of gameplay, game drawing and building, and keyboard interaction. ”BuildModel()” is the class that we use to initialize the background, dropped blocks and current blocks (drawables), in addition to taking in the traced object's position. ”BuildView()” is the class responsible for updating the view of the game as dependent on the state of ”BuildModel()” and its components. ”Build()” is the main run class; the class that initializes ”BuildModel()”, ”BuildView()” and ”PygameKeyboardController()”. It uses ”BuildModel()” to create the environment and take the tracer's outputs, ”BuildView()” to update the model, and ”PygameKeyboardController()” to receive user input. Other than creating and managing our game environment, we use the class ”PygameKeyboardController()” to respond to user's keyboard input.

The class ”DrawableSurface()” is a class called in ”Block()” that wraps a pygame.Surface and a pygame.Rect. ”Background()” is a class that simply represents the size attributes of the environment of our game. ”Background()” is called inside of ”BuildModel()” since it is part of the environment of our game, and ”DrawableSurface()” is used to get the drawables for our blocks.

For a full description of our classes, see the class diagram on the last page of this reflection.

In designing our game, one of the challenges we decided to try to implement was having collision detection in our game. We decided to initialize each block with a rectangle drawn around it, the dimensions of which would be the dimensions of the imaged block. We would then use pygame's built-in collision code to test whether or not the current block's rectangle collided with any of the past block's rectangles. We were able to successfully test collision within the ”Block()” class itself, but when we tried to connect collision detection to the keyboard input (the idea being that collision was checked every time the space bar is pressed) we ran into bugs with our code. We met with a ninja and saw that too much of our code would need to be reworked to implement collision successfully, and decided that with our time constraints it wasn't worth it. Additionally, we felt extremely successful with the number of goals we had already achieved in the game, as well as our personal learning goals.

4 Reflection

We started off our project by working on the gaming component and camera-tracking component separately. Based on our personal learning goals, we each took a set of tasks to complete and integrated them once they were done. From this point on, we pair-programmed and worked as a team apart from the final stages (cleaning up code, for instance) that were less critical to learning and could be done more individually. This worked extremely well from a coding and a learning perspective.

As partners, our work habits are extremely similar: we like to get work done well before deadline and consult ninjas as soon as a bug is deemed ”unsolvable” so as not to waste time. From this perspective, we were very successful in getting work done in a way that maximized efficiency and learning.

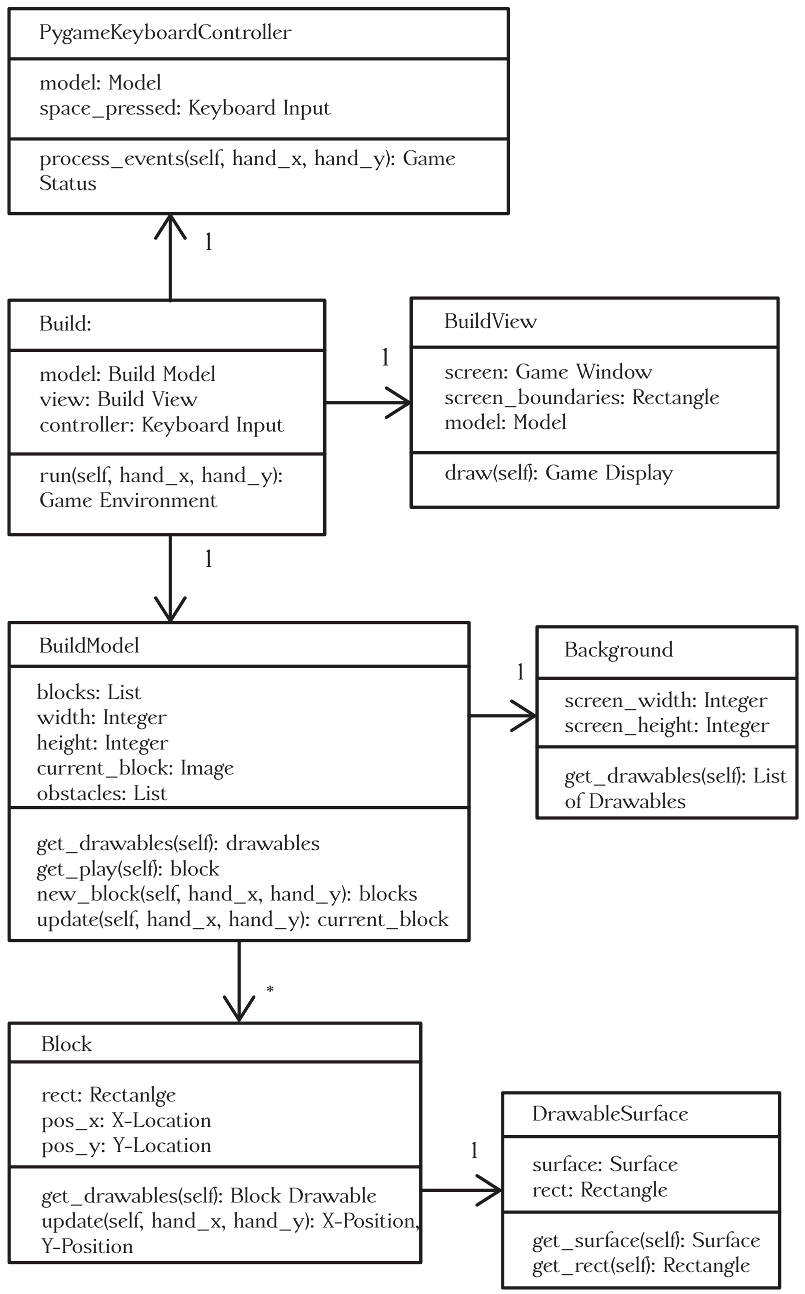
PygameKeyboardController
model: Model
space_pressed: Keyboard Input
process_events(self, hand_x, hand_y): Game Status
1
Build:
model: Build Model
view: Build View
controller: Keyboard Input
run(self, hand_x, hand_y): Game Environment
1
BuildView
screen: Game Window
screen_boundaries: Rectangle
model: Model
draw(self): Game Display
1
BuildModel
blocks: List
width: Integer
height: Integer
current_block: Image
obstacles: List
get_drawables(self): drawables
get_play(self): block
new_block(self, hand_x, hand_y): blocks
update(self, hand_x, hand_y): current_block
1
Background
screen_width: Integer
screen_height: Integer
get_drawables(self): List of Drawables
*
Block
rect: Rectanlge
pos_x: X-Location
pos_y: Y-Location
get_drawables(self): Block Drawable
update(self, hand_x, hand_y): X-Position, Y-Position
1
DrawableSurface
surface: Surface
rect: Rectangle
get_surface(self): Surface
get_rect(self): Rectangle

Figure 2