An Interactive Customization and Animation Program for American Football Offensive and Defensive Play Design

Michael C. Levecque
CIS4914, Senior Project
Advisor: Dr. Rong Zhang, *email*: rzhang@cise.ufl.edu
Date of Talk: 5 Dec 2018

Abstract

Recent developments in analytics, advanced sport statistics, and technology has caused a surge in sensationalism surrounding the numbers behind America's most popular sports. As a result of non-transparent and powerful analytic strategies being labeled proprietary and guarded at the risk of losing a competitive edge, a growing disparity has emerged between the majority of players and spectators with limited access to these tools and the ruling class of professional players and team/league executives.

While professional football is completely immersed in advanced analytics and related technological platforms, there has been a lack of major advancements for general use that could help developing players, coaches, and fans further conceptualize and understand the game. The aim of the project presented in this report was to develop a solution that benefited these non-professional stakeholders in the football community.

In response to this, a program was developed – **PlayMate –** to provide an immersive system based in low-level analytics, simplified statistics, and streamlined behavioral algorithms where users can design custom football play spreads for both offense and defense and then run animations of the interactions of their custom matchups.

1. Introduction.

With the increased use of advanced statistics, analytics, and visual models behind closed doors across modern American professional sports, it has resulted in a disconnect between the elite minority class of professional stakeholders and the common majority class of players and fans. In professional baseball, since the emergence of sabermetrics in the late 1970s, analytics have boomed to the forefront of organizational approach by teams in the Major League Baseball (MLB). Notable successes in analytics include the popular "Moneyball" story of the Oakland Athletics' unprecedented 20-game win streak in 2002, as well as the Boston Red Sox in 2004 snapping an 86-year World Series drought. Focusing on American football, advanced statistic and analytic platforms have been readily integrated into the National Football League (NFL) to help conceptualize the game, predict outcomes, and strategize game plans. Apart from the palettes of statistics that plaster screens during games and news outlets like ESPN, the NFL has seen the inclusion of instant-video replay/review, use of Microsoft Surface tablets on the sidelines, and adoption of advanced platforms like Next Gen Stats just within the past five years.

Unfortunately, as professionals at the highest level adopt these complex, protected analytics and platforms, non-professional levels of the football community have fallen behind. The solution to this problem, as outlined in this report, was the development of a system that integrates low-level statistical, behavioral, and algorithmic approaches in a comprehensive program that provides the user a visual tool to conceptualize and understand the game of football through play spread design and animation. This solution serves as an accessible platform that bridges the gap to the high-lofting analytic approaches used by professionals, provides a means to further exploration of the game, and logically extends visualization models that coaches and players have used for decades.

1.1. Problem Domain.

As technology and computer science become increasingly influential within modern society, cultural aspects such as sports naturally evolve as well. With the consistent obsession in the sports world for optimized athletic, statistic, and competitive performance, it is apparent that sports strategy has evolved into a computer science concern. The solution in this report relates particularly to the object-oriented, data abstraction, algorithm, and visual programming areas within computer science. By abstracting low-level statistics into custom player objects, the solution system is responsive to user interaction and outputs visual results through handling abstracted components deemed central to play design in football. Clearly, the solution also deals with user-experience design and human-centered computing areas. The resulting solution program makes all abstracted data accessible to the user. Providing such handling allows for the user to “touch” all the components during use, and thus further conceptualize and understand these considerations as they relate to football strategy. Further, the solution relates to other areas such as simulated behavior, and object interaction. Finally, informally related to areas of psychology, the solution expands on traditional visual models used for teaching and learning football plays.

1.2. Literature Search

The first task in the execution of the project was to research. The focus of the research three-tiered. First, NFL historical statistics (Pro Football Reference) were consulted to get a general feel of the variety and types of data recorded, as well as to start determining which major variables would be used in custom class and algorithm design. Football strategy and game theory videos (NFL 101) were also viewed to refresh personal knowledge and potentially offer further design insight. Second, on a technical level, it was ensured to have a reliable resource on retainer to help cross-reference proper coding procedures, conventions, and structure practices specific to Java-based graphics applications (Haase). Third, algorithmically, inspiration was drawn from a resource on natural phenomena that could be modeled programmatically (Flake). With topics like random directed walk algorithms, automata behavior and Turtle Graphics (Flake), it was hoped that these ideas could be extended to apply within the developed solution. Finally, a handbook on a separate, Java-based visual programming platform was consulted for simplified versions of collision, follow, and other related algorithms that could prove useful for implementing Non-Playable Character (NPC) type behavior to custom Player objects (Reas).

2. Solution.

In response to the issue of an increasing disconnect between the professional-level and developing non-professional level in game conceptualization and understanding in American football, the program **PlayMate** was developed. **PlayMate** is a Java-based graphical application that highlights abstracted Player objects, low-level statistics and data for play design, simplified behavioral algorithms for animation, and an accessible GUI for unlimited user manipulation. Ultimately, this application offers the target users (non-professional players and coaches, general football fans) a single platform to design, customize, and simulate offensive and defensive play spreads and their interactions. Currently, the application solely exists as a project within the IntelliJ IDEA environment and has not been compiled into a downloadable form.

Using an object-oriented programming language such as Java was critical to the solution because it allowed for the design of custom Player objects and classes whose attributes are carefully organized, managed and manipulated throughout the entirety of the

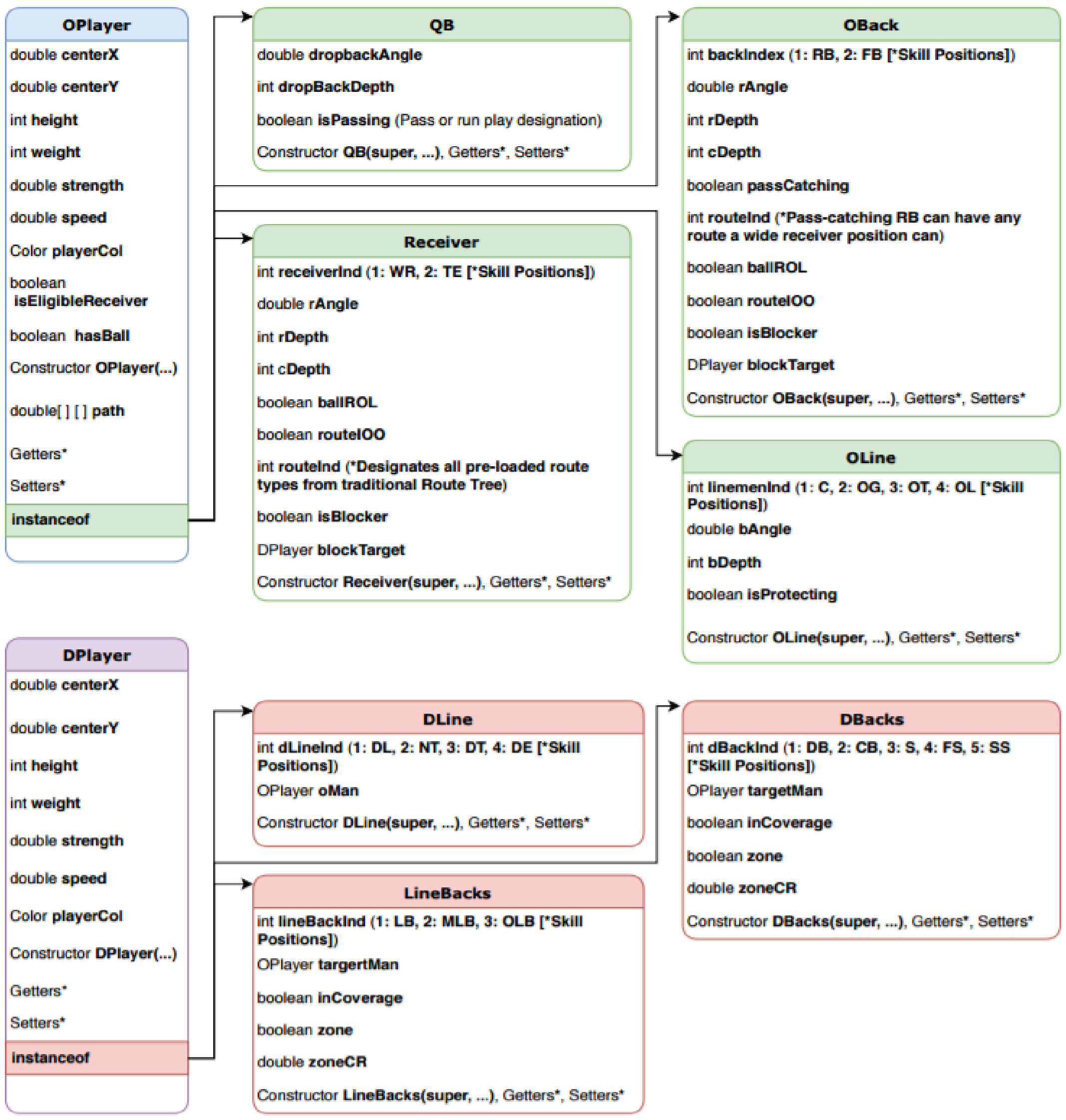

OPlayer
double centerX
double centerY
int height
int weight
double strength
double speed
Color playerCol
boolean isEligibleReceiver
boolean hasBall
Constructor OPlayer(...)
double[ ] [ ] path
Getters*
Setters*
instanceof
QB
double dropbackAngle
int dropBackDepth
boolean isPassing (Pass or run play designation)
Constructor QB(super, ...), Getters*, Setters*
Receiver
int receiverInd (1: WR, 2: TE [*Skill Positions])
double rAngle
int rDepth
int cDepth
boolean ballROL
boolean routeIOO
int routeInd (*Designates all pre-loaded route types from traditional Route Tree)
boolean isBlocker
DPlayer blockTarget
Constructor Receiver(super, ...), Getters*, Setters*
OBack
int backIndex (1: RB, 2: FB [*Skill Positions])
double rAngle
int rDepth
int cDepth
boolean passCatching
int routeInd (*Pass-catching RB can have any route a wide receiver position can)
boolean ballROL
boolean routeIOO
boolean isBlocker
DPlayer blockTarget
Constructor OBack(super, ...), Getters*, Setters*
OLine
int linemenInd (1: C, 2: OG, 3: OT, 4: OL [*Skill Positions])
double bAngle
int bDepth
boolean isProtecting
Constructor OLine(super, ...), Getters*, Setters*
DPlayer
double centerX
double centerY
int height
int weight
double strength
double speed
Color playerCol
Constructor DPlayer(...)
Getters*
Setters*
instanceof
DLine
int dLineInd (1: DL, 2: NT, 3: DT, 4: DE [*Skill Positions])
OPlayer oMan
Constructor DLine(super, ...), Getters*, Setters*
LineBacks
int lineBackInd (1: LB, 2: MLB, 3: OLB [*Skill Positions])
OPlayer targertMan
boolean inCoverage
boolean zone
double zoneCR
Constructor LineBacks(super, ...), Getters*, Setters*
DBacks
int dBackInd (1: DB, 2: CB, 3: S, 4: FS, 5: SS [*Skill Positions])
OPlayer targetMan
boolean inCoverage
boolean zone
double zoneCR
Constructor DBacks(super, ...), Getters*, Setters*

Figure 1: Class Diagram for parent OPlayer and DPlayer classes and their respective child subclasses.

program. The crux of the program was developing two custom Player classes – one to describe a player as part of the offense (OPlayer) and the to describe a player as part of the defense (DPlayer). Both classes share common global attributes that were determined by the author to be the most general, important descriptors of physical stature and football performance (height, weight, strength, speed). Although these are redundancies when comparing the classes, it proved important and beneficial in development to create two separate player classes, as further variables served to distinguish defensive and offensive players apart (e.g., certain offensive players are assigned to run routes with the goal of receiving an open pass, while defensive players often have different coverage assignments). After the general constructor layer, each of the player classes were broken down in child instances. To avoid unnecessary redundancies in creating a child subclass for every skill position the subclass design was simplified to reflect accepted football theory. Traditionally, both offensive and defensive players fall into one of three levels: 1) the Backfield, 2) the Line, and 3) the Receiver/Secondary Corps. Thus, subclasses were designed to reflect these levels, as skill positions within the same level perform similar actions or have similar attributes. What resulted were parent classes with respective child subclasses defined with all respective variables, constructors, getters, and setters, and simultaneously can be used to describe any skill position. The class breakdown diagram can be reviewed in **Figure 1.**

The programs structure employs basic Java Swing/AWT setup procedures to ensure proper initialization of application display protocols and Runnable and Thread queuing. Proper window buffering was confirmed throughout the project process and the program currently runs and responds in real-time, showing the correct execution of Runnables and Threads. The final program constitutes a single-JFrame, multi-JPanel structure. Conceptually, it was standard to initialize the overall application through an extension of JFrame within an overall custom ImageFrame class structure. Two separate extensions of JPanel were instantiated (**playSimulationPanel** and **interfacePanel**), as well. A **playSimultation** panel, named **mainPanel**, was directly linked to a BufferedImage with a corresponding Graphics2D representation where visual output components of play spread design and animation could be constantly painted to, displayed, and updated. On the other hand, multiple **interfacePanel** JPanels were designated for the different UI panels containing static and responsive control elements. **Figure 2** below shows a representation of the java graphics application structure for the solution.

The two major functionality sets of **PlayMate** are complete offensive and defensive play spread customization and animation of the resulting spreads. Both sets, algorithmically, include components that are precalculated and determined in real-time. Regarding play design, the most notable components that are pre-calculated are the passing routes ran by Receiver and pass-catching OBack OPlayer object instances. When an OPlayer registers states that they are an available receiver option, the program calls for the

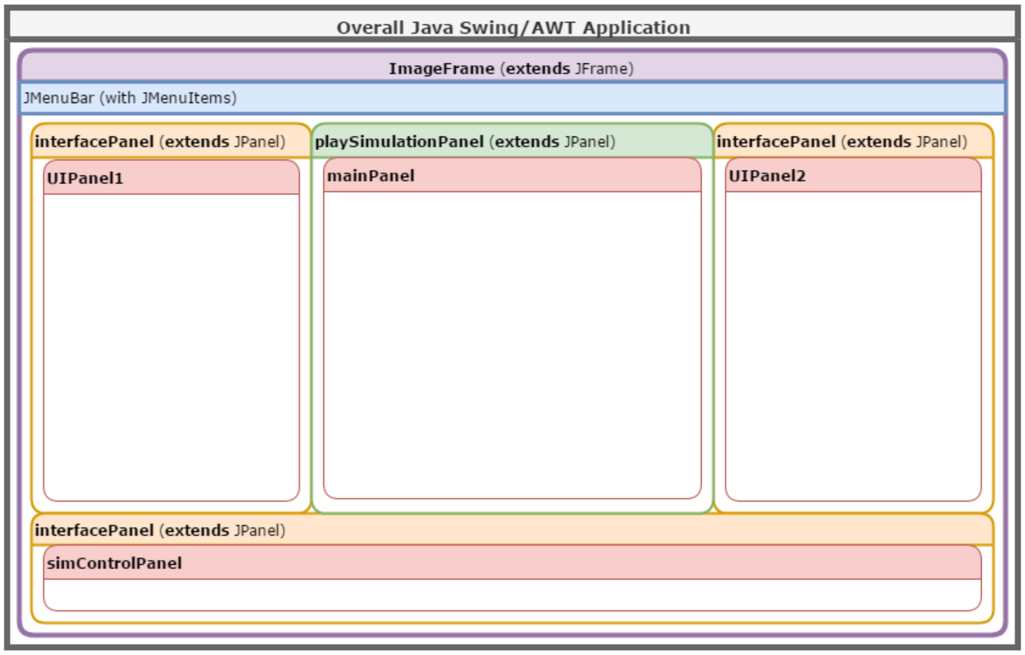
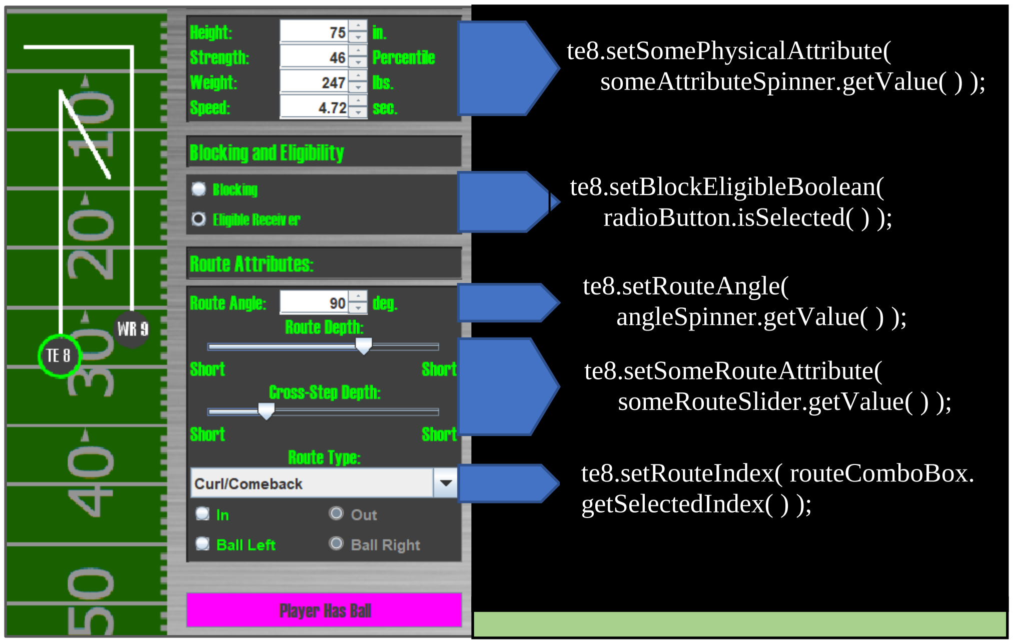

Overall Java Swing/AWT Application
ImageFrame (extends JFrame)
JMenuBar (with JMenuItems)
interfacePanel (extends JPanel)
UIPanel1
playSimulationPanel (extends JPanel)
mainPanel
interfacePanel (extends JPanel)
UIPanel2
interfacePanel (extends JPanel)
simControlPanel

Figure 2: Graphical Java Application structure of solution program.

OPlayer objects route index variable, calculates the entire route originating at the OPlayer's center x- and y-coordinates, and then draws it to the play design panel. Calculation of the different route types is handled in methods inspired by directed random walk algorithms (Flake). The animation engine primarily relies on reactive, real-time next-step movement calculation with condition checks. Recognizing that in the real-world there may be dozens of potential collisions for any given play on the football field, the animation algorithm is based on collision detection. When a collision is registered for any given step in the frame count between any combination of two player class types, procedures are taken to decide whether player objects move out of each other's way (DPlayer/DPlayer, OPlayer/OPlayer) or if a winner of a matchup needs to be determined and then subsequent movement action is taken (OPlayer/DPlayer). A pseudocode algorithm for both the collision method (**Appendix A**) and simulation engine (**Appendix B**) are provided.

Finally, the GUI for **PlayMate** was designed to allow the user to manipulate and customize all internal variables that influence the program and provide an organized, logical layout of all available controls. All available controls fall under three main classifications – the entire offensive or defensive spread, an individual player, or running the animation. Each data control classification has its own main panel to make the program organization more intuitive. A detail screenshot of the Player Attribute control panel and how it actively updates the selected OPlayer attributes is shown here:



Height: 75 in.
Strength: 46 Percentile
Weight: 247 lbs.
Speed: 4.72 sec.
Route Angle: 90 deg.
Route Type: Curl/Comeback

```
te8.setSomePhysicalAttribute(
    someAttributeSpinner.getValue( ) );

te8.setBlockEligibleBoolean(
    radioButton.isSelected( ) );

te8.setRouteAngle(
    angleSpinner.getValue( ) );

te8.setSomeRouteAttribute(
    someRouteSlider.getValue( ) );

te8.setRouteIndex( routeComboBox.
getSelectedIndex( ) );
```

Figure 3: Control Panel Detail and Pseudocode Showing the ActionListener Response of Java GUI Components and How They Update Class Variables Throughout Program

3. Results

The developed **PlayMate** program tested successful for all implemented functionality. In terms of must-have functionality for play spread design and customization, all player objects and their respective routes (OPlayer), blocking assignments (OPlayer), and coverage assignments (DPlayer) properly register from stored player object attributes and display to the design panel. All player object icons are draggable and all associated visual elements update responsively and actively in real-time. All user GUI controls in the three separate panels function as expected and actively update the associated attributes linked to the selected player for manipulation. Regarding the animation functionality set, the simulation engine produces sufficient movement, behavior, and interactions between player objects of uniquely customized and set play spread matchups. Because **PlayMate** is a graphical application, the testing and verification of the program functionality was carefully inspected manually. Similarly, there is not another appropriate technical test for the program other then direct visual testing. For proof of successful program testing, a demo video survey of the program functionality is linked within **Appendix C**.

4. Conclusions

Overall, a complete program was developed with a well-rounded functionality core, intuitive GUI, and comprehensive potential, as demonstrated from **Appendix C**.

After reviewing the current capabilities of the program and taking into consideration its potential to meet the needs of stakeholders at different levels in the football community, it is concluded that the solution at its completed state would be satisfactory for developing players and coaches leading up to and including those at the high school level. While the program boasts great customization and other aspects, the program does show limitations in meeting the needs for advanced or complex use. The advantages and disadvantages of the current program are summarized as follows:

Program Advantages	**Program Disadvantages**
• Great focus on route attributes and dynamic user manipulation. • Comprehensive visual representations of block and coverage assignments (zone radials, man-to-man coverage lines, nearest-defender blocking radials). • Simple play spread personnel management. • Responsive, draggable player icons. • Individual player position breakdown and ease of changing player objects. • Dynamic player physicality variables with direct effect on animation/simulation outcome. • Direct user access to play formation and individual player actions and behavioral states. • Satisfactory simulation quality and behavioral animation. • Animation controls and mid-simulation play customization available.	• Lack of complex potential for some skill positions (e.g. QB). • Lack of pass-rush route customization for DPlayer objects. • Lack of football intelligence/instinct factors in simulation engine. • Lack of complex simulation functions (setting key-frames on the frame line with associated animation actions or state-changes made by/to player objects). • Only pre-loaded formations available (user can't save a designed formation and load it later). • Lack of support for certain play types (e.g. QB designed runs).

Considering the above, as well as general review of the final program, future work would include, but not be limited to:

1. Expansion of OPlayer and DPlayer classes and subclasses to support complex plays.
2. Custom route design functions (control-point/interpolated routes) for all player objects.
3. Complex animation capabilities such as key-framing and automatic state changes.
4. Implementation of a probability-based play outcome decision engine. For example, the animation determining at the end if the play resulted in a completed pass, turnover, etc.
5. Extension to a web-application with a database and user profiles.

5. Standards and Constraints

Standards: All programming was done using Java (Platform SE 7), and specifically employing the Java Swing and Abstract Window Toolkit (AWT). All development was done using the IntelliJ IDEA, version 2017.3.4 x64, with JDK 9.0.4.

Constraints: All program components, methods, and GUI elements with their associated operations were developed to run responsively in real-time.

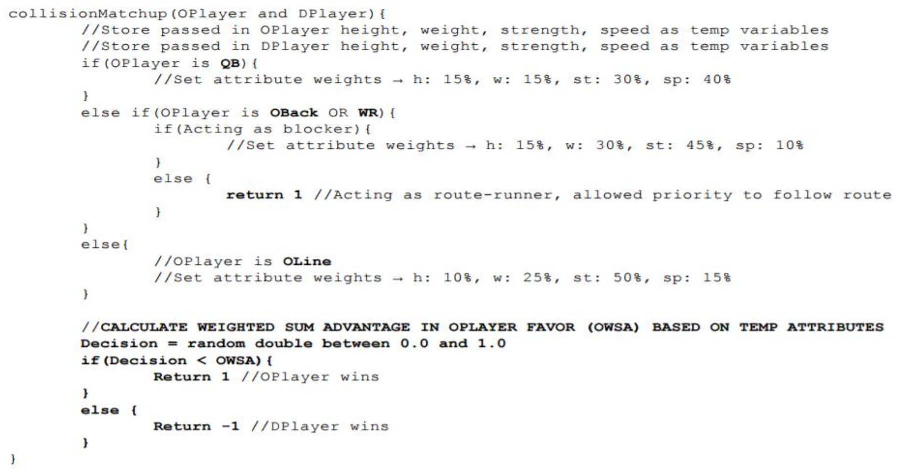
6. Acknowledgements

The author would like to thank his advisor, Dr. Rong Zhang, for her inspiration to him as a course professor, and for her guidance, advice, and consideration throughout. The author recognizes Dr. Zhang’s constant understanding and kindness as both professor and advisor, and credits her for the motivation for successful completion of this project.

7. References (MLA)

1. Flake, Gary William. The Computational Beauty of Nature. The MIT Press, 1998.
2. Haase, Chet, and Romain Guy. Filthy Rich Clients: Developing Animated and Graphical Effects for Desktop Java Applications. Addison-Wesley, 2007.
3. “Playlist - NFL 101.” YouTube, uploaded by NFL, 27 April, 2016, https://www.youtube.com/playlist?list=PLRdw3IjKY2gmCjwfEWnyY-QNxji0YSGaV
4. Reas, Casey, and Ben Fry. Processing: A Programming Handbook for Visual Designers and Artists. 2nd ed., The MIT Press, 2014.
5. Pro Football Reference: Football Stats and History. Sports Reference LLC, 2000, https://www.pro-football-reference.com/. Accessed 27 October 2018.

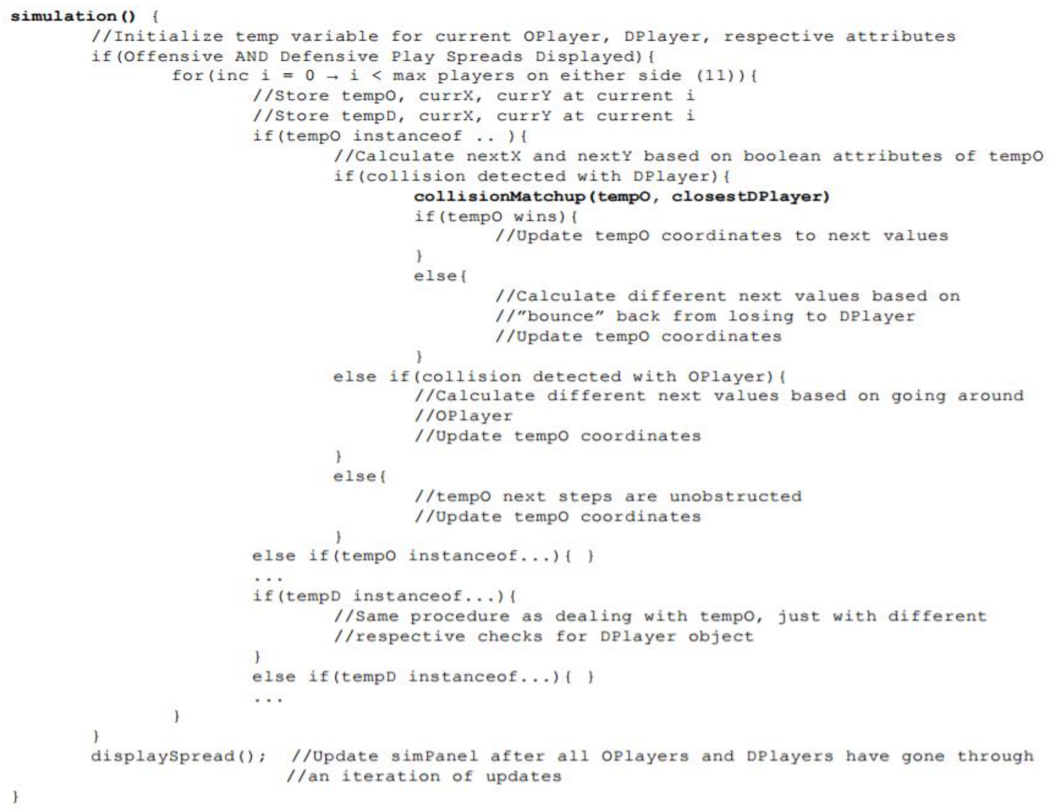
Appendix A



```
collisionMatchup(OPlayer and DPlayer){
        //Store passed in OPlayer height, weight, strength, speed as temp variables
        //Store passed in DPlayer height, weight, strength, speed as temp variables
        if(OPlayer is QB){
                //Set attribute weights → h: 15%, w: 15%, st: 30%, sp: 40%
        }
        else if(OPlayer is OBack OR WR){
                if(Acting as blocker){
                        //Set attribute weights → h: 15%, w: 30%, st: 45%, sp: 10%
                }
                else {
                        return 1 //Acting as route-runner, allowed priority to follow route
                }
        }
        else{
                //OPlayer is OLine
                //Set attribute weights → h: 10%, w: 25%, st: 50%, sp: 15%
        }

        //CALCULATE WEIGHTED SUM ADVANTAGE IN OPLAYER FAVOR (OWSA) BASED ON TEMP ATTRIBUTES
        Decision = random double between 0.0 and 1.0
        if(Decision < OWSA){
                Return 1 //OPlayer wins
        }
        else {
                Return -1 //DPlayer wins
        }
}
```

Appendix B



```
simulation() {
    //Initialize temp variable for current OPlayer, DPlayer, respective attributes
    if(Offensive AND Defensive Play Spreads Displayed){
        for(inc i = 0 → i < max players on either side (11)){
            //Store tempO, currX, currY at current i
            //Store tempD, currX, currY at current i
            if(tempO instanceof .. ){
                //Calculate nextX and nextY based on boolean attributes of tempO
                if(collision detected with DPlayer){
                    collisionMatchup(tempO, closestDPlayer)
                    if(tempO wins){
                        //Update tempO coordinates to next values
                    }
                    else{
                        //Calculate different next values based on
                        //"bounce" back from losing to DPlayer
                        //Update tempO coordinates
                    }
                else if(collision detected with OPlayer){
                    //Calculate different next values based on going around
                    //OPlayer
                    //Update tempO coordinates
                }
                else{
                    //tempO next steps are unobstructed
                    //Update tempO coordinates
                }
            else if(tempO instanceof...){ }
            ...
            if(tempD instanceof...){
                //Same procedure as dealing with tempO, just with different
                //respective checks for DPlayer object
            }
            else if(tempD instanceof...){ }
            ...
        }
    }
    displaySpread();  //Update simPanel after all OPlayers and DPlayers have gone through
                      //an iteration of updates
}
```

Appendix C

https://www.youtube.com/watch?v=7XJKgeNC9mM&feature=youtu.be

(Please use Microsoft Edge or Safari browsers)

8. Biography

Michael C. Levecque was born in Key West, Florida on November 21, 2018. As part of a military family, he attended five schools in three states and Puerto Rico until completing his secondary education in the International Baccalaureate program at St. Petersburg High School in St. Petersburg, Florida. He is currently completing his Bachelor of Science Digital Arts and Sciences degree from the Herbert Wertheim College of Engineering at the University of Florida, where he expects to graduate on December 15, 2018. Michael is a visual artist, painter, sculptor, and handyman when not programming for school. He hopes to incorporate his creativity and passion for art in his career. Upon graduation, Michael hopes to find a job in the area of his degree or plans to go to trade school for welding. Either way, maybe he will determine later that being a starving artist isn't all that bad, but at least there are several avenues available through his degree or trade to live a comfortable life while furthering his passion.