Dart Scoring System

Critical Design Review

EN.525.743 – Embedded Systems Development Laboratory

Kevin Parlak

Contents

[Figures 3](#_Toc146511651)

[Tables 3](#_Toc146511652)

[Project Description 4](#_Toc146511653)

[Problem 4](#_Toc146511654)

[Solution 4](#_Toc146511655)

[Capabilities 4](#_Toc146511656)

[Limitations 5](#_Toc146511657)

[Functional Description 6](#_Toc146511658)

[Imaging System 6](#_Toc146511659)

[State Machine 7](#_Toc146511660)

[Dartboard Mapping Logic 8](#_Toc146511661)

[Scoring System 9](#_Toc146511662)

[State Machine 9](#_Toc146511663)

[Gameplay 11](#_Toc146511664)

[Mobile App 13](#_Toc146511665)

[Interface Description 14](#_Toc146511666)

[Material Requirements 15](#_Toc146511667)

[Resource Requirements 16](#_Toc146511668)

[Development Plan and Schedule 18](#_Toc146511669)

[Milestones and Schedule 18](#_Toc146511670)

[Risk 19](#_Toc146511671)

[References 21](#_Toc146511672)

[Appendix 22](#_Toc146511673)

[Acronyms 22](#_Toc146511674)

# Figures

[Figure 1: Home Dartboard 4](#_Toc146566806)

[Figure 2: Functional Block Diagram 6](#_Toc146566807)

[Figure 3: Imaging System State Machine Diagram 7](#_Toc146566808)

[Figure 4: Dartboard Scoring 8](#_Toc146566809)

[Figure 5: Scoring System State Machine Diagram 9](#_Toc146566810)

[Figure 6: Scoring Framework 11](#_Toc146566811)

[Figure 7: "501" Logic 12](#_Toc146566812)

[Figure 8: “Around the World” Logic 13](#_Toc146566813)

[Figure 9: Roadmap Example 18](#_Toc146566814)

# Tables

[Table 1: Dartboard Mapping 9](#_Toc146566815)

[Table 2: Hosted Games 11](#_Toc146566816)

[Table 3: “Around the World” Mapping 13](#_Toc146566817)

[Table 4: Bill of Materials 15](#_Toc146566818)

[Table 5: VS Code Extensions 16](#_Toc146566819)

[Table 6: Python Libraries 16](#_Toc146566820)

[Table 7: Third-Party Applications 16](#_Toc146566821)

[Table 8: Milestone Schedule 19](#_Toc146566822)

[Table 9: Risk Impact/Probability 20](#_Toc146566823)

# Project Description

## Problem

Today, modern dartboards have built-in ways to automatically calculate user scores as players take turns throwing. However, traditional steel-tipped darts use cork dartboards which do not have a means of keeping score. This means the player must know how a specific game is scored and manually keep score as the game progresses.



Figure 1: Home Dartboard

## Solution

Dartboards are unique in that they are perfect circles and utilize different colored sections to indicate score location. This makes the game of darts a great candidate for using a computer vision solution to detect dart location. The dart scoring system solution will consist of multiple subsystems that interact to identify darts and score games automatically. The goal of this project is to produce the core capabilities needed to implement such a solution. Capabilities and limitations are defined that describe what will and will not be possible with the system being designed.

## Capabilities

The system will have the following capabilities to compliment core functionality:

* Scoring system database shall be capable of holding up to 10 player profiles and information
* User interface shall be capable of updating displays after each dart is thrown and identified
  + Virtualizes games for players through hit map/scoreboard updates
* User interface/mobile application shall be capable of pulling statistics such as win %, hit % by number, double ring hit %, triple ring hit %, outer bullseye hit %, inner bullseye hit % per profile
  + Helps players understand how their game is doing
* Imaging system shall be capable of re-calibration after each game
  + Allows dartboard imaging corrections to occur each time a game is played

## Limitations

System capabilities will be limited based on development time and resources available. The system will have the following limitations while ensuring core capability is achieved:

* Scoring system database shall only be accessible via home network or WAP
* Imaging system shall only be capable of tracking a single set of darts per turn
  + Requires players to select profile before throwing
* Scoring system shall only host two games (score-based and knockout-based)
  + Provides framework for adding future games
* System shall only be started from the physical user interface
* System shall only have a two-player limit per game
* Scoring system database shall only add new user profiles from the physical user interface
* Mobile application shall only be usable on Android devices

# Functional Description

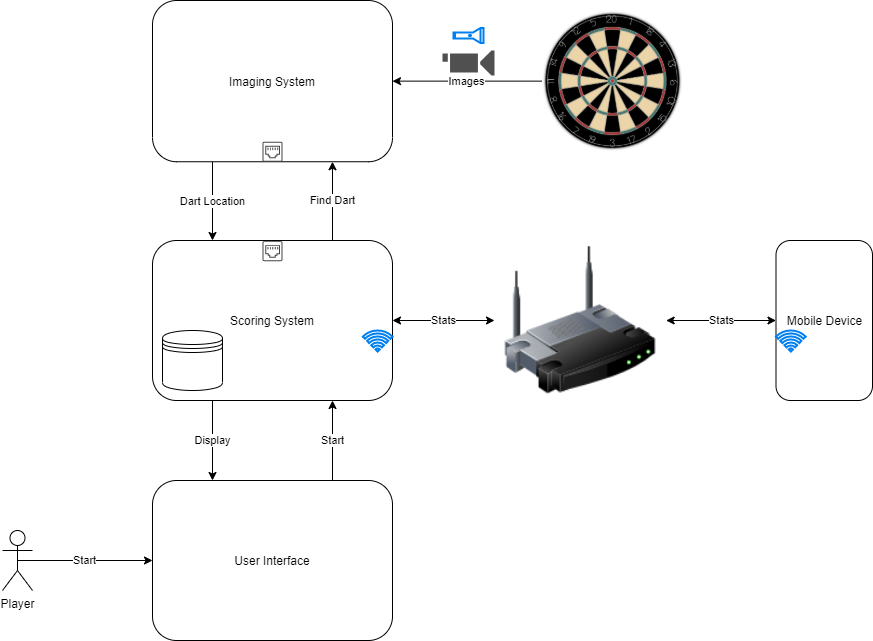


Figure 2: Functional Block Diagram

Figure 2 illustrates how the dart scoring system will work. The system will function as follows:

1. Player(s) will interact with the user interface to select their profile and a game of choice
2. Scoring system will load profiles and wait for player input
3. Before throwing, player will select their profile
4. Scoring system will indicate to imaging system a dart is incoming
5. Imaging system will start looking at dartboard, find dart, and report location back to scoring system
6. Scoring system will update game and player statistics from throw
7. Player(s) may query statistics from mobile app

This process repeats as players take turns throwing until a winner is declared or the game is ended.

## Imaging System

The imaging system is the subsystem responsible for identifying and locating darts on a dartboard.

### State Machine

A diagram of a system

Description automatically generated

Figure 3: Imaging System State Machine Diagram

Figure 3 shows how states will flow within the imaging system. This system will be comprised of image recognition logic. The imaging system state machine will function as follows:

1. After power-up the application will open and enter the WAIT START state. The system will remain in this state until the system receives a request to calibrate. After receiving the calibration request the system will enter the IDLE CALIBRATE state.

TODO : WAIT START flow diagram

1. The system will remain in the IDLE CALIBRATE state until proper calibration of the imaging system is complete.

TODO : IDLE CALIBRATE flow diagram

1. Once calibration is completed the system will enter the DONE CALIBRATE state. In this state the system will send a message to the scoring system that calibration is complete. The system will then enter the WAIT LOOK state.

TODO : DONE CALIBRATE flow diagram

1. The system will remain in the WAIT LOOK state until a message from the scoring system is received indicating a dart is incoming. After the message is received the system will enter the FIND DART state.

TODO : WAIT LOOK flow diagram

1. The system will remain in the FIND DART state until the system recognizes a dart has hit the dartboard.

TODO : FIND DART flow diagram

1. Once a dart is recognized the system will enter the MAP DART state. In this state the system will send a message to the scoring system with the coordinates of the dart that was recognized. The system will then enter the WAIT GAME state.

TODO : MAP DART flow diagram

1. The system will remain in the WAIT GAME state until a message from the scoring system is received indicating the status of the game being played. If the game is declared over, the system will enter the WAIT START state. If the game is still active, the system will enter the WAIT LOOK state.

TODO : WAIT GAME flow diagram

### Dartboard Mapping Logic

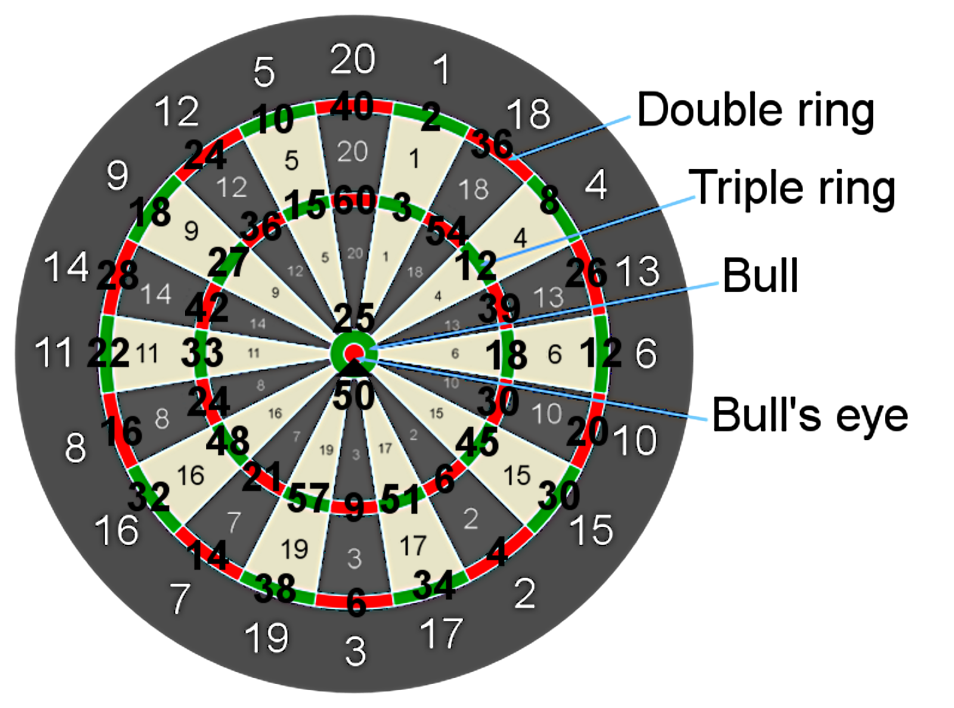


Figure 4: Dartboard Scoring

Figure 4 shows a dartboard map and associated raw scores. The imaging system will break each scoring piece into subsections to streamline game implementation. Each subsection will be broken into A, B, C, or D except for the bull, bullseye, and outside the dartboard.

|  |  |
| --- | --- |
| **Location** | **Ring Map** |
| Double ring | A |
| Between double and triple ring | B |
| Triple ring | C |
| Between triple ring and bull | D |
| Bull | X |
| Bullseye | Y |
| Outside dartboard | Z |

Table 1: Dartboard Mapping

Table 1 shows how each dartboard location will be mapped within the imaging system. For example, if a player hits the double ring of 20, the system will report value 20, ring A. If a player hits the bull the system will report value 0, ring X. If a player hits the bullseye the system will report value 0, ring Y. If a player hits outside the dartboard the system will report value 0, ring Z.

## Scoring System

The scoring system is the subsystem responsible for updating user scores based on game mode as well as controlling user interaction. The user interface is part of the scoring subsystem.

### State Machine

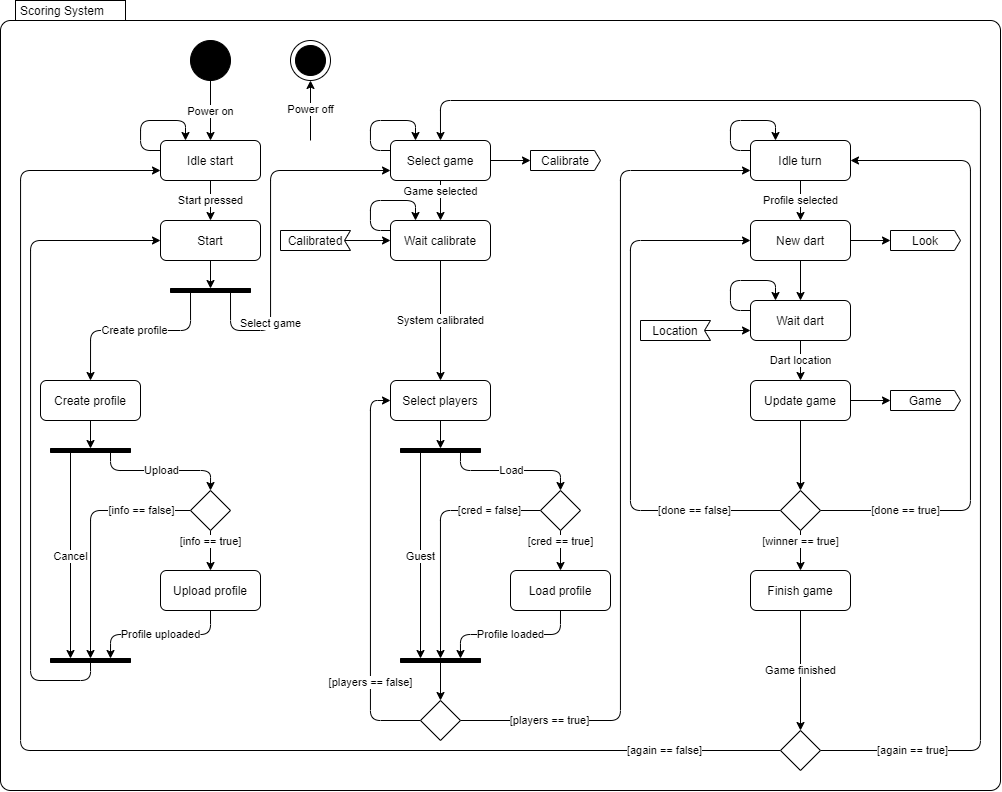


Figure 5: Scoring System State Machine Diagram

Figure 5 shows how states will flow within the scoring system. This subsystem will be comprised of game and user interface logic. The scoring system state machine will function as follows:

1. After power-up the application will open and enter the IDLE START state. The system will remain in this state until the player starts the application from the user interface. After starting the application, the system will enter the START state.

TODO : IDLE START flow diagram

1. The system will remain in the START state until the player creates a profile or selects a game.

TODO : START flow diagram

1. If the player creates a profile, the system will enter the CREATE PROFILE state. In this state, the player will enter information to upload or cancel the create request.

TODO : CREATE PROFILE flow diagram

* 1. Upon upload, the system will check to ensure information is filled out and enter the UPLOAD PROFILE state if valid. The system will remain in this state until data is uploaded to the database.

TODO : UPLOAD PROFILE flow diagram

1. If the player selects a game, the system will enter the SELECT GAME state. In this state, the system will send a request to the imaging system to calibrate itself and enter the WAIT CALIBRATE state.

TODO : SELECT GAME flow diagram

1. The system will remain in the WAIT CALIBRATE state until a message is received from the imaging system indicating that it is calibrated. Upon receipt of this message, the system will enter the SELECT PLAYERS state.

TODO : WAIT CALIBRATE flow diagram

1. The system will remain in the SELECT PLAYERS state until the appropriate number of players are ready for the game specified. In this state, the player will play as a guest or load an existing profile. After the appropriate number of players are selected, the system will enter the IDLE TURN state.

TODO : SELECT PLAYERS flow diagram

* 1. Upon upload, the system will check to ensure credentials are filled out and enter the LOAD PROFILE state if valid. The system will remain in this state until data is loaded from the database.

TODO : LOAD PROFILE flow diagram

1. The system will remain in the IDLE TURN state until the user selects their profile before throwing darts. Once a player profile is selected the system will enter the NEW DART state.

TODO : IDLE TURN flow diagram

1. In the NEW DART state, the system will send a request to the imaging system indicating a dart is incoming and enter the WAIT DART state.

TODO : NEW DART flow diagram

1. The system will remain in the WAIT DART state until a message is received from the imaging system indicating it has recognized and located a dart. Upon receipt of this message, the system will enter the UPDATE GAME state.

TODO : WAIT DART flow diagram

1. In the UPDATE GAME state, the system will update player throw statistics and scores based on the game selected. The system will also send a message to the imaging system of the state of the game. If the system calculates that the player won, the system will enter the FINISH GAME state. If a winner is not declared, the system will check two other conditionals. If the player has thrown the appropriate amount of darts per turn the system will enter the IDLE TURN state, otherwise the system will enter the NEW DART state.
2. In the FINISH GAME state all player data will be uploaded to the database. Once complete, the system will check if the player(s) want to play again. If yes, the system will enter the SELECT GAME state. If no, the system will enter the IDLE START state.

TODO : FINISH GAME flow

### Gameplay

|  |  |
| --- | --- |
| **Scoring Games** | **Knockout Games** |
| 501 | Around the World |

Table 2: Hosted Games

The system will be capable of hosting scoring and knockout games. Table 2 shows what games will be hosted from each category.

A diagram of a graph

Description automatically generated

Figure 6: Scoring Framework

Figure 6 shows the framework the system will use to score games. First, player scores will be loaded. Next, the CALCULATE SCORE state will be entered. Here, game specific logic will be followed for the incoming dart location. Once scores are calculated the system will update displays and exit the UPDATE GAME state.

#### “501” Game

The goal of “501” is to reach a score of 0. Players start with a score of 501 and count down as areas of the dartboard are hit. Players throw darts to any position, sum each throw, and subtract from 501. Double rings count as double multipliers while triple rings count as triple multipliers of each sections’ value. The bull counts as 25 points while the bullseye counts as 50 points. Players win by hitting the double ring to reach a score of 0 even.

Referencing Figure 4, if a player has 20 remaining, they must hit a double 10 to win. If they hit a double 5, they need a double 5 on their next throw to win.

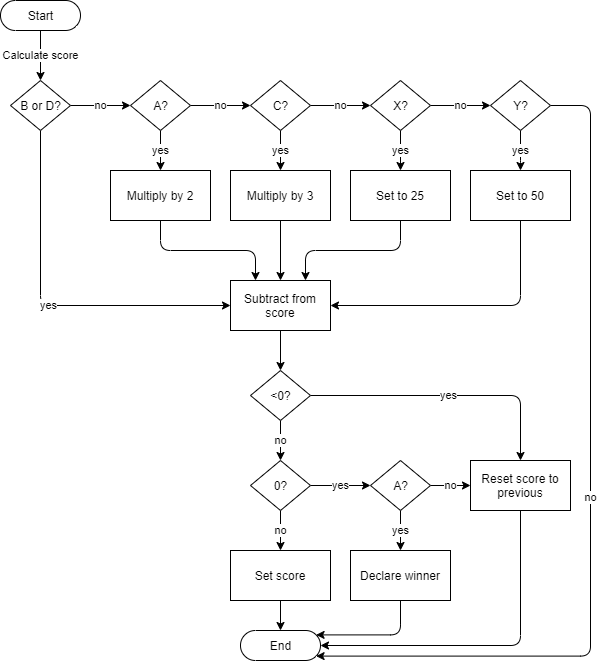


Figure 7: "501" Logic

The system will use mappings as seen in Table 1 to determine multipliers and scores to tally player scores throughout the game. Figure 7 shows how “501” will logically work. After entering the CALCULATE SCORE state, the system will check what section was reported by the imaging system. The section determination will yield a multiplier, straight value, or nothing. The system will then subtract the modified value from the loaded score. If the score goes below 0, the system will reset the score to the loaded value. If the score equates to 0, the system will check if a double ring was hit. If a double ring was hit, the system will declare a winner. Otherwise, the system will reset the score to the loaded value. If the score is above 0, the system will set the computed score.

|  |  |  |  |
| --- | --- | --- | --- |
| **ASSERT** | | | **EXPECT** |
| **Section** | **Ring** | **Current Score** |
| 20 | A | 501 | 461 |
| 20 | B | 501 | 481 |
| 20 | C | 501 | 441 |
| 20 | D | 501 | 461 |
| 20 | X | 501 | 476 |
| 20 | Y | 501 | 451 |
| 20 | Z | 501 | 501 |
| 20 | A | 20 | 20 |
| 20 | B | 20 | 20 |
| 10 | A | 20 | 0 |

Table 3: "501" Unit Testing

Table 3 shows how the game logic for “501” will be unit tested. These scenarios will test each leg of the logic chain to ensure bulletproof game logic.

#### “Around the World” Game

The goal of “Around the World” is to hit each section of the dartboard in sequential order. Players throw darts starting at position 1 and move clockwise around the board. Double and triple rings count as singles towards the section hit. The bull and bullseye count as nothing. Players win by knocking out all sections, finishing with 20.

Referencing Figure 4, if a player hits 1, 4, 18, they knockout 1 but are only on 4 because 18 was the last sequential section hit.

|  |  |
| --- | --- |
| **Current Section** | **Next Section (A, B, C, or D)** |
| 0 | 1 |
| 1 | 18 |
| 18 | 4 |
| 4 | 13 |
| 13 | 6 |
| 6 | 10 |
| 10 | 15 |
| 15 | 2 |
| 2 | 17 |
| 17 | 3 |
| 3 | 19 |
| 19 | 7 |
| 7 | 16 |
| 16 | 8 |
| 8 | 11 |
| 11 | 14 |
| 14 | 9 |
| 9 | 12 |
| 12 | 5 |
| 5 | 20 - WINNER |

Table 4: “Around the World” Mapping

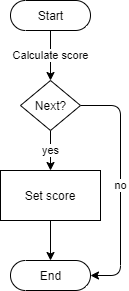


Figure 8: “Around the World” Logic

The system will use a mapping technique to check the current and next section to determine sequential knockout of the dartboard. Figure 8 shows how “Around the World” will logically work. After entering the CALCULATE SCORE state, the system will check the section reported by the imaging section. This value will be graded against the next section associated with the current section as seen in Table 3. If the values equate, the system will set the current section to the new section.

## Mobile App

# Interface Description

# Material Requirements

The dart scoring system will be comprised of two main computing systems.

The scoring system will run on a Raspberry Pi 3B+. This microprocessor has built-in Wi-Fi capability in addition to Ethernet communication. The Raspberry Pi also has CSI connectivity which will be used to communicate with a touchscreen user interface.

The imaging system will run on a Nvidia Jetson Nano Developer Kit. This GPU-based microprocessor has built-in Ethernet communication as well as CSI connectivity which will be used to communicate with an external camera.

|  |  |  |
| --- | --- | --- |
| **Item** | **Price** | **Website** |
| Raspberry Pi 3B+ | $35.00 | https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/ |
| SanDisk Ultra 64GB MicroSD Card | $8.99 | https://www.amazon.com/SanDisk-Ultra-microSDHC-Memory-Adapter/dp/B08GYBBBBH |
| FREENOVE 5 Inch Touchscreen | $39.95 | https://www.amazon.com/FREENOVE-Touchscreen-Raspberry-Capacitive-Driver-Free/dp/B0B455LDKH?th=1 |
| Raspberry Pi Power Supply | $9.95 | https://www.amazon.com/CanaKit-Raspberry-Supply-Adapter-Listed/dp/B00MARDJZ4 |
| Nvidia Jetson Nano Developer Kit | $149.00 | https://developer.nvidia.com/embedded/jetson-nano-developer-kit |
| SanDisk Ultra 64GB MicroSD Card | $8.99 | https://www.amazon.com/SanDisk-Ultra-microSDHC-Memory-Adapter/dp/B08GYBBBBH |
| Jetson Nano Metal Case w/ Fan | $25.59 | https://www.amazon.com/Metal-Case-Fan-Jetson-Nano/dp/B07Z2MFTYC?th=1 |
| SainSmart IMX219 8MP Camera Module | $21.99 | https://www.amazon.com/SainSmart-IMX219-Camera-Module-Raspberry/dp/B07VFFRX4C |
| Jetson Nano Power Supply | $13.68 | https://www.amazon.com/5V-Power-Supply-Adapter-Universal/dp/B07RTWD725?th=1 |
| **TOTAL** | $313.14 |  |

Table 5: Bill of Materials

# Resource Requirements

Development for the dart scoring system will occur in the Python language. Python was chosen due to the amount of FOSS libraries and examples that will be referenced during project development.

VS Code will be used to write Python code for both computing systems. This IDE was chosen because of the FOSS nature of the environment and due to the number of extensions possible for faster development within the Python language.

|  |  |
| --- | --- |
| **Extension** | **Reason** |
| Python | Rich support for language |
| Pylance | Static type checking tool |
| Intellicode | AI-assisted development tool |
| Remote Explorer | View remote machines for SSH |
| Remote-SSH | Open folders on remote machines using SSH |

Table 6: VS Code Extensions

Table 2 shows a list of VS Code extensions that will be used for faster system development. These extensions will allow remote development to occur on the Raspberry Pi and Jetson Nano processors.

|  |  |  |  |
| --- | --- | --- | --- |
| **System** | **Reason** | **Library** | **Documentation** |
| Scoring | Database development | sqllite3 | https://docs.python.org/3/library/sqlite3.html |
| Scoring | GUI development | pyqt5 | https://pypi.org/project/PyQt5/ |
| Scoring | Graphics | turtle | https://docs.python.org/3/library/turtle.html |
| Scoring | TCP/IP communication | socket | https://docs.python.org/3/library/socket.html |
| Imaging | Computer vision | opencv-python | https://pypi.org/project/opencv-python/ |
| Imaging | Machine learning | tensorflow | https://pypi.org/project/tensorflow/ |
| Imaging | TCP/IP communication | socket | https://docs.python.org/3/library/socket.html |

Table 7: Python Libraries

Table 3 shows a list of Python libraries that will be used to implement the dart scoring system. These libraries will accelerate design implementations.

Android Studio will be used to develop the mobile application for the dart scoring system. This IDE was chosen because it streamlines GUI development and testing for Android applications all in one software package.

|  |  |
| --- | --- |
| **Application** | **Reason** |
| VNC Viewer | Remote Raspberry Pi/Jetson Nano access |
| Wireshark | TCP/IP packet observation |

Table 8: Third-Party Applications

Table 4 shows a list of third-party applications that will be used during development. These applications will be used as necessary for debugging purposes.

Existing dart scoring projects will be referenced to speed up development and integration of this project.

The following projects will be referenced for image processing and dart detection:

* <https://developer.nvidia.com/embedded/community/jetson-projects/dart_score>
* <https://github.com/hanneshoettinger/opencv-steel-darts>
* <https://github.com/wmcnally/deep-darts>

Standard dart rules will be referenced for implementation of the scoring system:

* <https://dartsguide.net/guides/dart-games-rules/>

# Development Plan and Schedule

Planning and tracking of the dart scoring system will take place in GitHub. Code will undergo CM in an online repository which can be accessed from the following link:

<https://github.com/kparlak/dart-scoring-system>

GitHub Projects will be used to plan and track progress throughout development. Projects allow milestones to be tracked with related issues. Issues within Projects will be tied to merge requests on the main repository. The online project can be accessed from the following link:

<https://github.com/users/kparlak/projects/2>



Figure 9: Roadmap Example

Figure 6 shows a snapshot of the project roadmap in GitHub. Milestones are represented as vertical lines traversing the plan. Issues will be given start and end dates within GitHub to align progress with milestone and completion dates.

## Milestones and Schedule

This project will use milestone-based development, meaning a new capability will be introduced when a milestone is completed. Unit testing will be used to drive individual milestone capabilities. This will ensure each subsystem is functional part and apart from the overall system.

The first milestone will be Dart Recognition Development. The goal of this milestone will be to implement computer vision and image processing on the imaging system. After this milestone is complete the system will be able to do the following:

* Recognize a dart
* Transmit dart location via TCP/IP

The second milestone will be Scoring and Game Development. The goal of this milestone will be to implement game logic for “501” and “Around the World” on the scoring system as well as communication with the imaging system. Building on the previous milestone, after this milestone is complete the system will be able to do the following:

* Receive dart location via TCP/IP
* Calculate score based on game being played
* Determine game winner

The third milestone will be Database Development. The goal of this milestone will be to implement a database on the scoring system and store data from games being played. Building on the previous milestone, after this milestone is complete the system will be able to do the following:

* Host database for user profiles and scores
* Upload data to database as game progresses

The fourth milestone will be User Interface Development. The goal of this milestone will be to implement user interface designs on the scoring system. This includes hit maps and scoreboards as well as game flow interfaces. Building on the previous milestone, after this milestone is complete the system will be able to do the following:

* Enable user interaction with touchscreen interface
* Display scoreboard and hit maps

The full system will be functional following the fourth milestone. Mobile App Development will be the final milestone but capabilities will run in parallel with the dart scoring system. The goal of this milestone will be to implement an Android app that will be able to query the hosted database for player statistics and scores. Building on the previous milestone, after this milestone is complete the system will be able to do the following:

* Use app to view statistics

|  |  |
| --- | --- |
| **Date** | **Milestone** |
| 9/25 – 10/16 | Dart Recognition Development |
| 10/16 – 10/30 | Scoring and Game Development |
| 10/30 – 11/6 | Database Development |
| 11/6 – 11/20 | User Interface Development |
| 11/20 – 12/4 | Mobile App Development |

Table 9: Milestone Schedule

## Risk

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Low | Impact | | | High |
| Very Likely |  |  |  |  |  |
| Likelihood |  | **1** |  |  |  |
|  |  |  | **2** |  |
|  |  |  |  | **3** |
| Not Likely |  |  |  |  |  |

Table 10: Risk Impact/Probability

Table 9 shows the impact/probability table for risks associated with this project. The current risks for this project are as follows:

1. Loss of home internet connectivity

Mitigation: Switch Raspberry Pi to act as WAP rather than Wi-Fi client when internet connectivity is lost

1. Machine learning for image recognition likely to be difficult

Mitigation: Three weeks built-in for image recognition milestone; use known trained datasets from online repositories to feed solution

1. Consistent lighting and angle viewpoints for imaging system likely non-deterministic

Mitigation: Implement calibration routines that will run before games are played

# References

<https://developer.nvidia.com/embedded/jetson-nano-developer-kit>

<https://www.raspberrypi.com/news/coding-on-raspberry-pi-remotely-with-visual-studio-code/>

# Appendix

## Acronyms

|  |  |
| --- | --- |
| CM | Configuration Management |
| CSI | Camera Serial Interface |
| FOSS | Free and Open Source Software |
| IDE | Integrated Development Environment |
| SSH | Secure Shell |
| VS | Visual Studio |
| WAP | Wireless Access Point |