ECE 2564

Project 1

The Little Explorer

Summer 2021

(V2 - 6/15/2021)

**How do I read this document?**

Our goal has been to provide you with as much information and tools as possible to help you succeed with the project. Nevertheless, you might not need to read the entire document at first. We recommend that you take a quick look at it to know the main topics covered. Then, carefully read the [Game Description](https://docs.google.com/document/d/1v7KTzXlz_fbGmTMGXATCUMIaAIXNwl5OMuSNc-6DT_w/edit#heading=h.nzbfi64fgu4y) and [Code Quality](https://docs.google.com/document/d/1v7KTzXlz_fbGmTMGXATCUMIaAIXNwl5OMuSNc-6DT_w/edit#heading=h.naky4uckjjg4) sections, and begin your design. Once you have made some progress with your project, carefully study the [Validation](https://docs.google.com/document/d/1v7KTzXlz_fbGmTMGXATCUMIaAIXNwl5OMuSNc-6DT_w/edit#heading=h.najxomm3xqzj) section. Whenever you have any questions, you should first try to find your answer in this document.

**Table of Contents**

**(Essential sections of this document are marked with an asterisk \*.)**

[Fundamentals](#_hanjj68mhf97)

[General guidelines for any assignment](#_5mc0npgkj2m7)

[Why do I have to complete this project?](#_dbrvf039tbx7)

[Starter Code](#_4asj43v99bzv)

[Honor Code Policy](#_m5da7te6ibba)

[Golden Solution](#_9brqjrejqi37)

[Progress checksheet](#_gsxocoidy6ib)

[Submission Process\*](#_nyhfipcwmiya)

[Button/LED Names](#_ml8nbftljdmu)

[Game Description\*](#_j0i0haj74izn)

[The Story](#_y3iot7bdawu7)

[Summary of the Game Stats](#_xgdxuyukad83)

[Playing the Game via Terminal](#_qce33avhulis)

[Game’s Avatar and the Play Area](#_v8x3cvalmp91)

[Title (Splash) Screen and Game-Over Screen](#_2n4e5ybtxvam)

[Game Progression](#_a3ye1ns8rezu)

[Response to the Terminal](#_tvl65dynlosq)

[UART Baud Rates](#_lmq3tmwfbtp)

[Summary of the Requirements](#_4sdlab277i5h)

[Code Quality\*](#_naky4uckjjg4)

[Comments](#_dikdoyf8dmlp)

[No Global Variables](#_dypype5xl1wa)

[No Numeric Values](#_gbaznj38onjk)

[No Long Functions](#_2aao9443pbjx)

[Using HAL](#_ecd24y27h7rg)

[Non-Blocking Code](#_e4l153lectb5)

[Bonus Point Opportunities](#_5cac1xxoqnz9)

[Bonus Points Grading](#_u0f4po9lt5jl)

[Design Strategy](#_2me7iycqa5n5)

[Report](#_kuq47r6e1bed)

[Validation\*](#_najxomm3xqzj)

## 

## Fundamentals

### General guidelines for any assignment

* Before you start working on your assignment:
  + Skim through the entire project description and carefully read the essential sections marked with \*
  + Read the starter code carefully and in its entirety.
  + Run the provided \*.out on your board. This is an executable that we call the ***golden solution***.
* Every time you sit down to work on your assignment:
  + Before you start coding, **go to Piazza, and read questions/notes related to the assignment you are working on**, ***even if you took a two-hour break***.
  + After you finish coding, push your changes to GitHub, ***even if you are going to take a two-hour break***!

### 

### Why do I have to complete this project?

TI’s MSP432P401R Launchpad is an inexpensive microcontroller which has a variety of peripherals which can be used for many applications. The purpose of this assignment is for you to gain experience with UART as well as to explore designing a larger project on the MSP432P401R. You will develop a “Hangman” game that is displayed on the board and controlled through a UART connection with your computer keyboard. This project will develop your skills in implementing FSMs, decomposing and developing a larger project, and developing non-blocking code for embedded systems.

### Starter Code

To get the starter code, follow the instructions in the first two steps of the CCS/GitHub integration from [​here](https://sites.google.com/vt.edu/introduction-to-embeddedsystem/tutorials/github-integration)​. The only difference is that instead of the assignment link in the tutorial, you need to use the below link:

[**https://classroom.github.com/a/7d5cdKoZ**](https://classroom.github.com/a/7d5cdKoZ)

### Honor Code Policy

You are allowed to copy code from any code given to you as part of the course material (HW, class examples, course website, etc.) In the Project Explorer, you can copy paste files or entire folders, such as the HAL folder in Su21\_HW6. **However, copying/inheriting code from outside/external resources constitutes plagiarism.** If you have any questions if your research or design falls under “fair use”, do not hesitate to contact the Professor, a GTA, or a ULA.

### Golden Solution

We are providing a golden solution executable. The purpose of this file is for you to observe the behavior of the project in action. The executable is called **Su21\_Project1\_golden.out** ​and is provided on Canvas. Follow the instructions ​[here](https://sites.google.com/vt.edu/introduction-to-embeddedsystem/tutorials/project-development/flashing?authuser=0) ​to flash the microcontroller with the executable file. ***If there is any difference between the golden solution and the project description, please follow the project description. Please also bring it up on Piazza so that we are aware of it.***

### 

### Progress checksheet

The below document provides a progress checklist that you can print and put somewhere that you can look at on a daily basis.

<https://docs.google.com/document/d/1G75k-K6VqulRzJh--g6420Y8WvVwDpYsgVO-DQSOoQg/edit?usp=sharing>

This checklist is not a substitute for the validation table which is embedded later in this document. Instead, this is a set of simple steps that can build toward completing the project. ***You do not need to complete these steps in the proposed order.*** This table is probably very useful for those of you having difficulty breaking a large assignment into smaller steps. Feel free to create your own version of the to-do list based on your own strategy.

### Submission Process\*

You will have to submit your project to GitHub by the deadline. ***It is extremely important that you make sure you have uploaded all your files, including custom images if any, to GitHub.*** You can use simple methods like counting the number of files in your local folder versus the files on GitHub. There are also more reliable methods to check your submission. You can learn more about this process here:

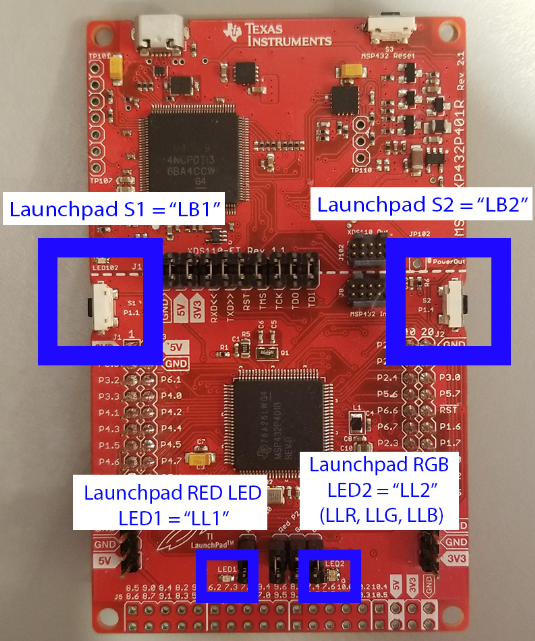
[***https://sites.google.com/vt.edu/introduction-to-embeddedsystem/tutorials/github-integration/verification-of-submission***](https://sites.google.com/vt.edu/introduction-to-embeddedsystem/tutorials/github-integration/verification-of-submission)

You are going to submit your report to Canvas as a pdf.

***If for any reason, you are not able to submit your work to GitHub, zip up your work and email it to your GTA and instructor.*** Regardless of the method, you will have to submit your work before the deadline.

### Button/LED Names

In this Project, we will use the LEDs and buttons on both boards. We will use the same naming convention as our previous assignments as shown below. For the BoosterPack LEDs, we will call the collection of all three LEDS, BLED which can be turned into a variety of colors.



## 

## Game Description\*

### The Story

Any good game tells a story. This game is the story of an Explorer who ***Moves*** around in their world as long as they have ***Energy***. As they move, they gain ***Experience***. When their experience reaches a predetermined threshold (***To Next***), they rise to a new ***Level***. Once they’ve leveled up, the Explorer’s experience becomes zero to reflect lack of experience in this new “world/level”. Moving decreases the explorer’s energy and feeding increases it. Each feeding provides one unit of energy, but one unit of energy can be used for several moves. Based on this story, the Explorer will have a: Level, Energy, Experience, Movement, and To Next stats responsible for controlling the progression of the game’s state.

### Summary of the Game Stats

The following table summarizes the game stats:

|  |  |
| --- | --- |
| **Stat** | **Description** |
| Level | Explorer’s level |
| Energy | Explorer’s energy |
| Experience | Explorer’s experience (in this current level) |
| To Next | Required experience to pass the current level  (this number only changes when the level increases) |
| Move | The number of moves the explorer has made (this number resets every time the energy level decreases) |

### Playing the Game via Terminal

The player controls the explorer mostly through commands that he/she sends to the board via UART connection and MobaXterm terminal. These commands are as follows:

|  |  |
| --- | --- |
| **Input** | **Command Operation** |
| w | Move Up |
| a | Move Left |
| s | Move Down |
| d | Move Right |
| f | Feed |

Whenever *any* input is received by UART, **including characters that don’t represent commands** (such as ‘q’, ‘r’, or ‘z’), the received character should be **echoed back** to MobaXterm. For example, typing ‘f’ into MobaXterm should also display ‘f’ on the serial terminal.

### Game’s Avatar and the Play Area

The basic “Explorer” can be as simple as a circle or rectangle. However, you are welcome to design more complicated avatars. You can even change your avatar as it enters higher levels or gains more experience. The play area in which the Explorer moves should be represented by a rectangle or some sort of border. Your Explorer should be able to move to at least **nine (9)** different positions within its borders. You can use a **3x3 grid of positions** or free-form moving.

### Title (Splash) Screen and Game-Over Screen

Like all great games, your game will have a **title screen**, which should contain the project name as well as your name. You are welcome to include other interesting graphics on this screen. Upon sending any key through UART, the game begins. When the Explorer dies, a **Game-Over** screen is displayed which includes printing the highest level achieved in gameplay.

### Game Progression

In order to **Move**, the Explorer must have at least **one (1) Energy** present to **Move**. **Energy** is a value within the inclusive range [0, 5]. **Energy** may not be less than zero and must not exceed 5. For every **two (2) Moves, one (1) Energy should be deducted**. The **Move** parameter on-screen reflects the number of moves since the last energy deduction. The Explorer may only **Move** into an **open area**. Sending a command telling the Explorer to **Move** into a border should **not** decrement the **Energy**, increment **Experience** or **Level**, and should not change the Explorer’s position.

If the **Energy** ever reaches **zero (0)**, it is **Game Over** and the **Game-Over Screen** should be shown. The Game-Over Screen should reflect the highest Level achieved.Once this screen is reached, the only way to reset the game is to press the RESET button on your board. *Note: The non-blocking LB1 check which turns on LL1 should still work here.w*

The Explorer starts out at **Level 0**. As your game progresses, each movement should increase **Experience** by **one (1)**. The amount of **Experience** required to **Level Up** increases with each **Level**. The formula we will use is *twice the current* ***Level****, plus 1*, or{(2 \* current) + 1}. For example, at Level 0, it takes {(2 \* 0) + 1} = 1 experience to **Level Up** the Explorer. At Level 5, it takes {(2 \* 5) + 1} = 11 **Experience** to **Level Up** the Explorer. **To Next** reflects the new amount of **Experience** required to **Level Up** from the previous level.

### UART Baud Rates

The UART Connection may have one of four speeds (BPS): 9600, 19200, 38400, 57600. On reset, the program must start at 9600 BPS. Tapping BB1 should perform a circular increment through the UART speeds. When configuring UART, use 8 data bits, one stop bit, and no parity bits.

Additionally, the BLED (Boosterpack RGB\_LED) should change color according to the current UART speed. The speed-color pairings are as follows:

|  |  |
| --- | --- |
| **Baud Rate** | **Boosterpack RGB LED Color** |
| BAUD\_9600 | Red |
| BAUD\_19200 | Green |
| BAUD\_38400 | Blue |
| BAUD\_57600 | White |

The UART implementation is easily represented as an FSM of the respective Baud Rate states shown in the table above. Note that in C when an FSM is represented as an ENUM, each state can be expressed as an integer starting at 0. Thus for the table above mapped as an ENUM:

* BAUD\_9600 -> 0
* BAUD\_19200 -> 1
* BAUD\_38400 -> 2
* BAUD\_57600 -> 3

(This information is provided as an aid for implementing the circular logic and toggling the UART Baud Rate speeds. You are NOT expected to display the Baud Rate on the screen, but you ARE required to illuminate the correct BLED color for the respective Baud Rate according to the above table).

## 

## Summary of the Requirements\*

For more information on the exact baseline requirements of your Basic Explorer, consult the following table.

|  |  |
| --- | --- |
| Title Screen | * A title screen is shown with the title of the game (your choice) and your name. * An instruction similar to this is shown: “Press any key on Mobaxterm to start” * A keystroke on Mobaxterm ends the title screen. |
| Display | * Experience (Exp) appears in the top left corner. * Level appears underneath Experience in top left corner. * Energy appears at the top right corner. * To Next appears underneath Energy in the top right corner. * Movement (Move) appears in the bottom right corner. * A border is drawn to the screen (and doesn’t overwrite characters) |
| Explorer | * Explorer appears on the screen. * Explorer appears within the borders. * Explorer is at least a circle or a rectangle with a diameter/width of 10 pixels. |
| Initial State | * Energy is 1. * Experience is 0. * Level is 0. * To Next is 1. * Baud is BAUD\_9600. * Move is 0. * BLR is ON |
| User Input Actions | * Pressing BB1 toggles the Baud Rate. It cycles through the values of 9600, 19200, 36400, and 57600. * Sending an ‘f’ over UART increases Energy by 1 (up to 5). * Sending ‘w’, ‘a’, ‘s’, or ‘d’ over UART moves the explorer to the up, left, down, or right respectively.   + The explorer only moves if there is open space to move in that direction.   + All movements should be contained within borders.   + If Energy is 0, the game ends and GAME OVER screen shows.   + Any movement which is blocked by borders should NOT consume energy, should NOT increase experience, should NOT increment the move counter, and should NOT end the game.   + Each valid movement increases Experience by 1.   + Two valid movements decrease Energy by 1. To keep track of this, the Movement variable shows the number of moves since the last change in energy level. |
| Game Over | * Only shown once Energy is 0. * Displays the Level achieved. * Pushing LB1 still causes LL1 to turn on. * No other input received from buttons or UART. * RESET correctly restarts the game back to initial state. |
| Leveling Up | * The explorer starts at level 0. * The threshold (To Next) for each level is determined by the formula {(2 \* current level) + 1}. * When the experience gained meets the threshold, the explorer levels up. For example, suppose explorer’s current level is 6. The threshold for the next level would be (2 \* 6) + 1 = 13. The explorer should advance from level 6 to level 7, and the experience should reset to zero. |

## Code Quality\*

Developing high-quality code is a very important skill for any programmer. A well-written program can be easily tested, updated and shared with the rest of the team, whereas a low-quality program might “work” for a short period of time but will not be practical in the long run. In this course, you will strive to develop higher quality code to build your skills. We will focus on the below aspects for this purpose.

### Comments

You should add enough comments to your code that makes it easy to read and understand. If you are not sure what level of documentation is enough, consult your instructor or your GTAs.

### No Global Variables

Your code should not use global variables. All interactions between functions should use parameter passing. The only exception *may* be a custom image (if you implement one).

### No Numeric Values

Do not use any numeric values inside your code with the exception of 1 and 0. Instead of hardcoding values in your code, use macros to associate required values with symbolic labels. This rule does not apply to the graphic functions.

### No Long Functions

All functions including the main function should be less than 50 lines of code. This is to encourage you to build a modular design with smaller functions that have simple, clear objectives. Comment lines, lines with only a single curly brace, and empty lines do not contribute to this 50 count. Points will only be deducted for serious violations of this rule. If a function is very close to the requirement (52, 53 lines, etc) that will be acceptable. **If a single function is above 60 lines of code (with the exception of provided libraries like grlib.h and ImageReformer) you will receive a 0/30 for this score.**

### Using HAL

You need to have a HAL for all your low-level hardware interactions including buttons and LEDs. You can use your code from any previous homework or project for this purpose.

### Non-Blocking Code

Your entire application should be non-blocking: At any point during the operation of the game, pressing LB1 should turn on LL1. Releasing LB1 should turn off LL1. This operation is to make sure none of your code is blocking the processor from running in an infinite loop responding to user input similar to a typical bare-metal embedded system.

## Bonus Point Opportunities

There are many opportunities to get extra points for this project. The bonus points are capped at **200 bonus points**. You are free to pick some features from the below list or discuss new features with the instructor.

* **Custom Explorer (75 points possible)**- Your Explorer changes physical appearance based on its Level. You choose the level thresholds (i.e. Levels: 1, 5, 7, etc). There are two point options available for this implementation based on difficulty
  + **(25/75 Points) -** Using a circle or rectangle to represent the Explorer, change the color of the Explorer depending on the Level. You must include at least 3 levels in a table with a description and the corresponding colors associated with the levels and include it in the report under the \*BONUS\* section.
  + **(75/75 Points) -** Use Image Reformer for custom graphics to represent the Explorer. You must include at least 3 different graphics (one for each level threshold) in a table with a description, and include it in the report under the \*BONUS\* section.
* **Custom Title and Game-Over Screen** **(25 Points)** - Using Image Reformer, make a custom Game-Over and Title Screen and include it in your report under the \*BONUS\* section.
* **LED color change based on Energy (10 Points):** (*Implemented in the Golden Solution, but not required.)* The Launchpad LED2 “LL2” is illuminated according to the energy level. The colors we have implemented are in the table below

|  |  |
| --- | --- |
| **Energy Level** | **Launchpad LED 2 Color (LL2)** |
| 5 | BLUE |
| 4 | TURQUOISE |
| 3 | GREEN |
| 2 | YELLOW / ORANGE |
| 1 | RED |
| 0 | OFF (DARK) |

* **Experience Bar** **(40 Points)**- At the top of the LCD, implement a progress bar (a rectangle) that increases across the screen according to the amount of Experience. This bar should **scale**, according to how much experience you have accumulated for the current level
  + At Level 1, Exp 1 -> Bar is ½ of screen
  + At Level 2, Exp 2 -> Bar is ⅖ of screen (notice how the denominator is calculated)
* **Unique Experience (100 points):** Change your code such that the experience is increased only if the explorer visits a new location in the play area (unique experience). This can only work for the first few levels where the To Next is low. You can add visuals to the play area that marks any location that is visited by the explorer to keep track of the visited locations.
* **Command output to MobaXTerm (25 points):** (*Implemented in the Golden Solution, but not required.)* Whenever a command sent through UART, output a character to MobaXTerm based on the type of command. Output ‘M’ for movement commands (‘w’, ‘a’, ‘s’, and ‘d’), ‘F’ for feeding commands (‘f’), and ‘U’ for any other character received.
* **Restarting the Game (30 points):** Similar to the Title screen, any keystroke from Mobaxterm on the Game-Over screen restarts the game.

You are welcome to implement any other extension you are interested in. **However, you need to discuss it with your instructor and get an email confirmation of what you are intending to implement is acceptable and how many points it is worth.**

### Bonus Points Grading

*The amount of bonus points received is a function of the amount of points received from the code-quality section (Rounded down). In other words….*

*Say you received 200/200 Bonus points, but only 70/100 Code-quality points. Thus, your resulting bonus points received would be:*

*== 0.70 ( 200 ) → 140 Final Bonus Points*

## Design Strategy

This project will require more effort than previous assignments. It may take several dedicated days, if not the whole two weeks to complete, especially if you struggled on the previous homework assignments or struggle with programming in C in general. Based on our experience in the past, we highly recommend you start projects in this course as soon as possible. **READ THE GAME SPECIFICATION IN ITS ENTIRETY AS SOON AS YOU CAN, so that you can start thinking about design issues you may encounter in your project sooner rather than later!**

In designing Project 1, we recommend the following strategy:

1. Get your peripherals to work (I/O and Hardware Interfacing)
2. Design your application-level software on paper
3. Implement each feature delineated in your design outline, one at a time
4. Use the debugging techniques we learned in earlier assignments: breakpoints, viewing variables, using LEDs, ...

As is always the case with any project involving code, always get your inputs and outputs working first before you work on the rest of the code. Import code from those previous examples, and **ensure that your peripherals still behave as expected**.

New to this project is the **UART module**. This module is unique in that its primary setup involves slightly more effort than previous peripherals. In configuring the UART module, there are plenty of additional considerations to make, including:

1. Understanding Driverlib. There’s an entire section on UART programming from this [link](https://sites.google.com/vt.edu/introduction-to-embeddedsystem/uart/programming-with-uart) on our course website. One of the simplest things you can do is to have your microcontroller echo any keypresses you have sent through MobaXTerm.
2. Design a UART module in a style similar to the <HAL/Button.h> and <HAL/SWTimer.h> HAL modules. Make sure to instantiate your UART module in <HAL/HAL.h> - **don’t be afraid to modify any code we give you! There’s always ways to elegantly extend your design while maintaining the abstraction layer principles we’ve explored in this class.**
3. When testing your UART module, test on **a single baud rate first.** For most of you, this is 9600 baud. Make sure this mode works before you expand this to other baud rates.

To test your peripherals, make simple helper functions which validate output as a function of input. The choice for how to do this is ultimately yours, but one simple example includes writing functions to **toggle launchpad LEDs when you receive button presses, or toggle boosterpack LEDs when you receive select characters from the UART terminal.**

NOTE: These are simply testing functions. Make sure you either comment them out or otherwise remove them when you move on to the next design step. **You do NOT need to implement any of the bullets directly above this note to receive full points on this assignment.**

Next, put away your code and get out a pen and some paper. The best software is designed with a pen and paper, not by hacking away with no stops until it barely works! Reread the project description and start jotting down some ideas for how you can implement parts of your project. Key components you may wish to consider in your design include the following:

* What kind of data do you need to represent your Explorer? How can you represent things like Movement, Energy, and Levels? How should you represent the current Baud rate?
* How can a player interact with your Explorer? What kind of inputs can they enact? How does player input influence the Explorer’s state/position? Drawing some FSMs on your paper might be useful to you. The C standard library also has plenty of string functions which are premade for you - doing some research might prove fruitful.
* How will you draw each UI element onto your LCD? Draw a mock-up on your paper to get a good sense of what the final product should look like, and how each state variable in the Explorer is represented on the LCD.
* **When** do you print an update to the screen? The LCD on your Boosterpack is slow. You cannot afford to clear and redraw everything on the LCD for every pass of your super-loop. You need to be more efficient than that. Try to keep your redraws to the screen to a minimum to reduce lag in your code.

As always, when implementing your code, practice an incremental approach. ***Implement each feature you have outlined on your paper one at a time, and immediately test the new feature you have just added so that you can detect possible bugs early!*** For example, if you choose to implement experience first, don’t be concerned with drawing your avatar in the middle of the screen quite yet. Get your experience and leveling to interact with each other properly first, then worry about graphics.

Finally, when using Git, use your commits wisely. We recommend committing your code after you successfully implement each new feature. This should seem obvious, but don’t commit broken code! If you know there is a bug somewhere, fix it before you commit. Each commit you make should be a checkpoint that you can roll your code back to if you really must.

## 

## Report

Write a short report with the following:

1. **Cover page:** This should include title, date, location, institution (Virginia Tech), and course number. ***You need to include your name and GitHub ID on the cover page as well.*** You can choose the format, but it needs to be in a formal professional form.
2. **Report Summary:** In less than 50 words summarize the purpose and the content of the report.
3. **Project Description:** A short section (a paragraph or two) describing what your application can accomplish. Ideally, this matches the Project description. However, use your own words to describe it. State clearly which parts of the Project specification are satisfied by your implementation. If your application cannot perform all the tasks or performs some of them suboptimally, you need to mention them in this section. **Make sure to also provide visuals for what your Explorer looks like in a few states, so that your GTA knows what to look for when grading the Explorer.** For example, include a *screenshot at start*, after *one feeding*, *one movement*, *one level* *up*, and the *Game-Over screen*. Label these with descriptions to inform the GTA as to what is happening in each screenshot.
4. **Main FSM:** Draw and describe the main FSM of the Explorer as a high-level model. Draw the arcs, describing the conditions that transition the FSM from state to state.

You do not have to try and define *every possible state* the Explorer can be in, but rather, derive a high-level block model or flowchart for the FSM requirements to interpret a command (move up, down, left, right, feed). *The goal here is for you to tell us, at a high level, what you have implemented. While you don’t need to necessarily outline every last permutation of every variable in your game, it is important to describe major state changes your Explorer undergoes as the game progresses. Ensure that your diagram conveys how commands are interpreted and how the private stats are internally updated.*

1. **Bonus:** Write one or more paragraphs explaining which bonus features your application achieves (if any), and how you implemented those. **Create a table or bulleted list with the features you have implemented and the number of points you expect to get from those bonus features. Add a row (or bullet) for the total points you expect to get from bonus points.**

When you have finished, submit your report to Canvas as a pdf.

## Validation\*

Your GTA will evaluate this project using the actions as described in the table below. Please ensure that your program follows the specification and that it passes the validation tests as best as possible locally before submitting!

|  |  |
| --- | --- |
| Feature | Point Values and Description |
| Basics  [200] | Reset the board. Open a 9600 Baud Rate terminal, with 8 data bits, one stop bit, and no parity bit.  [25] A splash screen is displayed.  [25] Sending any character via UART clears the splash screen and starts the game.  [25] An Explorer circle or square is displayed.  [25] Borders are displayed.  [25] Level, Energy, Experience, To Next, and Move meters are displayed in correct positions.  [25] Level, Energy, Experience, To Next, and Move initial states are correct. Verify that the list below matches the data on the LCD. Each status is worth [5] points.   * Level = 0 * Energy = 1 * Experience = 0 * To Next = 1 * Move = 0   Move the Explorer around to all positions.  [25] There are at least **nine (9)** unique positions.  [25] The Explorer doesn’t move outside borders. |
| Short Life  [200] | Reset the board. Open a 9600 Baud Rate terminal, with 8 data bits, one stop bit, and no parity bit.  [50] Explorer starts in the center of the screen, within borders and with correct initial states as specified earlier in this specification.  [50] Move in any direction once. Afterwards, the game state should be:   * Experience = 0 * Level = 1 * Energy = 1 * To Next = 3 * Move = 1   [50] Move into a wall. Nothing should happen or change.  [50] Move in any other valid direction. The Explorer should die, and the death screen should be shown. |
| Long Life  [400] | Reset the board. Open a 9600 Baud Rate terminal, with 8 data bits, one stop bit, and no parity bit. Play around with the Explorer. Verify that each feature below works properly.  Features based on Level:  [25] Level increases when {Exp == To Next}  Features based on Energy:  [25] Energy increases each time ‘f’ is received via UART.  [25] Energy is bounded to the range of [0,5]  [25] Energy is decreased every **two (2)** moves.  [25] Energy is **not** decreased if moving into a wall.  [25] On **zero (0)** Energy the game ends.  Features based on Experience:  [25] Every valid move should give **one (1)** experience.  [25] Each time the Explorer Levels Up, Experience should reset to **zero (0)**.  Features based on To Next:  [25] The required Experience to Level Up follows the formula  = {(2 \* current level) + 1}  [25] To Next accurately updates each Level  Features based on Move (movement):  [25] Explorer is able to reach all **nine (9)** positions of the playable area.  [25] Explorer is **unable** to advance beyond the borders.  Features based on baud rates:  [20] Baud rate performs circular increase whenever BB1 is pressed.  [20] BLED Colors light up correctly when Baud rate changes, with point subdivisions as follows:  [5] BLED = RED -> BAUD\_9600  [5] BLED = GREEN -> BAUD\_19200  [5] BLED = BLUE -> BAUD\_38400  [5] BLED = WHITE -> BAUD\_57600  [20] The tests above work when the baud rate is set to 19200 baud.  [20] The tests above work when the baud rate is set to 38400 baud.  [20] The tests above work when the baud rate is set to 57600 baud. |
| Code Quality  [100] | [10] Code is reasonably commented so that a TA can understand your code.  [10] #defines and constants are used instead of hard-coded numbers.  [10] No function exceeds 50 lines of code, within reason. Comment lines, lines with only a single curly brace, and empty lines do not contribute to this total. Points will only be deducted for egregious violations of this rule. If a function is very close to the requirement (52, 53 lines, etc) then that is also okay.  [20] No global variables are used, with the exception of possibly custom images using the Image Reformer.  [50] Your code is completely non-blocking. Pressing LB1 lights up Launchpad LL1 at any time. |
| Report  [100] | [10] No grammatical or spelling errors are present.  [10] Cover Page  [10] Report Summary  [30] Project Description  [40] Main FSM or high-level flowchart, pseudo code |
| Bonus  [200] | Report any bonus features and record how many bonus points were earned. In your report, you are required to create a table or a bulleted list with a row for each bonus feature and an extra row dedicated to the sum of points you expect to get for bonus features. Also, late submissions do not qualify for bonus points.  Points received from Bonus use the following formula:  Final Bonus = [Code Quality %] \* [Raw Bonus]  Final Bonus = 0 if submission is late. |
| Total w/o bonus  [1000] |  |