Software Documentation including (Requirement, design, test cases, and test results)

for

512 Parking Simulator

Version 0.3 (Final)

Prepared by 512 Parking Services

DATE: 04/28/2022

Table of Contents (2 pts)

Revision History (2 pts)

Name	Date	Reason for Changes	Version
Wade Nicolas	04/08/2022	Updated sections as per feedback	0.2
Sam Torbert	04/10/2022	Updated UML	0.2
Jesse Slager	04/12/2022	Updated sections as per feedback	0.2.1
Wade Nicolas	04/22/2022	Added in missing sections and Test Cases	0.2.2
Jesse Slager	4/27/22	Updated UML	0.2.3
Sam Torbet	4/28/22	Updated UML	0.2.4
Wade Nicolas	04/28/2022	Updated Test Cases, cleaned up final draft	0.2.5
Jesse Slager	04/28/2022	Cleaned up diagrams	0.2.6

1. Introduction

1.1 Purpose (5 pts)

The purpose of this project is to create a tool for businesses and organizations that allows them to estimate the amount of parking area, along with the appropriate lot locations required to maintain normal operations as well as an acceptable flow of traffic and personnel on a day to day basis. This tool will simulate the movement of different personnel and vehicles to different parking areas based on location and distance to an event.

1.2 Intended Audience and Reading Suggestions (2 pts)

The intended audience of this document would be the upper leadership of organizations (e.g. administration of universities) concerned with smoothing out the flow of traffic and people within their campuses and facilities. This software is intended for use at Embry Riddle Aeronautical University but can be expanded to other campuses and facilities if this project were to continue past the end of the semester. This would include Embry Riddle's sister campuses or any business/organization that expresses the desire to organize and obtain data about its parking situation.

1.3 Product Scope (3 pts)

As part of this deliverable, one can expect this project to contain a full color interactive map of the area, as well as information about each specific parking lot (Vehicle numbers, congestion, etc.). The project will also take into account variable conditions such as time of day and event location that may or may not affect how some people will park. The complete software package was originally going to be designed with modularity in mind. This includes configurable parking lot sizes, shapes, parking zone (color) changes, and spot variation. However, for the purposes of this deliverable, as well as the time constraints imposed upon it, this software will simulate the Embry-Riddle Aeronautical University parking lots, and will **not** allow for configurable parking lot sizes at this moment in time.

1.4 References (2 pts)

- ***Embry Riddle campus layout and parking lot layout.***
- ***Attempted to obtain list of vehicles on campus (denied by campus safety)***

2. Overall Description

2.1 Product Perspective (10 pts)

The software contained within the product will be a parking lot simulation that is designed to help consumers organize and plan their movements and for organizations to learn more about the parking patterns of their customers/employees. The complete software package will be designed with modularity as a focus. At the moment, the software will provide information on one specific location without the ability to switch locations. A partial dataset will be manually recorded with the intention of randomly generating the remainder of the dataset based on the same constraints found in the live examples. This should allow for a full dataset that is reminiscent of the student body without having to survey the entirety of the student body. The software will take into account and display the locations of vehicles within specified parking lot sections as well as display their color tags. Utilizing academic/residential buildings as points of interest and action, the software will run a simulation that attempts to find a suitable, open location where certain vehicles (depending on the color tag and the schedule of the driver) would or should park on campus at different times of day in order to reduce congestion and minimize the possibility of being late or severe inconvenience.

Figure 1. Context Diagram

The location in which a student or staff member would want to park can be influenced by multiple factors within the simulation. These factors include:

- Distance between the chosen parking lot and intended destination.
- Number of slots available in a given parking lot
- Time of arrival/departure
- Lot closures (Construction, events, etc.)
- Error (Conscious decision to park further)

For example, a student may arrive at 9:35AM and want to park in the Citation East Extension (CEE) parking lot, which is next to the Lehman building for a class at 9:45AM, however, as the lot is full, and there isn't much time left the student would choose the next available lot, which would be Citation Center (CC). However, if the student had arrived earlier, they may be willing to wait for a spot to open up in CEE. The answer to that decision making process is what the simulation will try to provide based on the data provided to it and the constraints.

2.2 Product Functions (5 pts)

The project will present the user with information about the designated parking area on a full-color interactive map. The user will then be able to select specific parking zones for more information and a detailed view of the vehicles (and their characteristics) within them down to the parking spot. The user will then be able to simulate certain events that would affect the behavior of personnel, such as time of day, class/work schedules, lot closures, weather, etc. to gain insight into how the aforementioned personnel go about their day.

2.3 User Classes and Characteristics (10 pts)

The primary users of this software would be organization/university administrations, with consumers/students as secondary users. Both types of users will be able to utilize the software in its entirety, however, users at the administrative-level for organizations would benefit from the ability to interact with parking lot permissions and events, as well as population density in order to test and evaluate different situations. The ability to rezone lots and alter the parking flow of an area would also assist in collecting data and making changes.

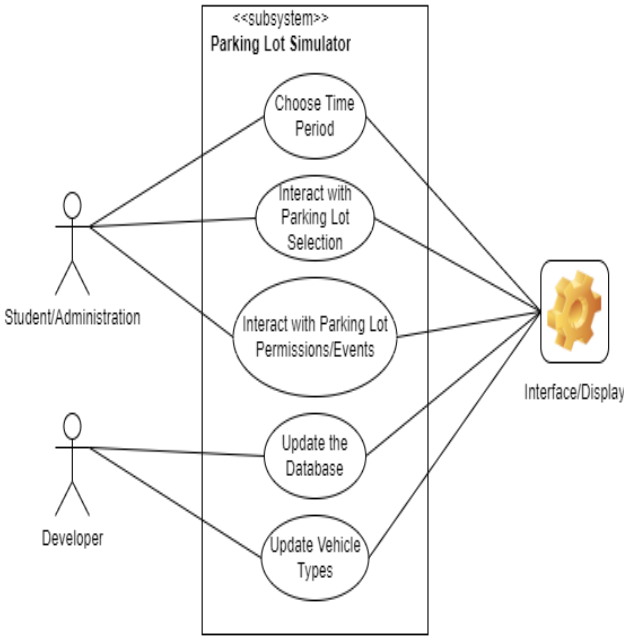


Figure 2. Use Case Diagram

Consumers, on the other hand, would benefit more from the ability to manipulate the time scale and desired destination in order to learn traffic flow patterns and the most effective times to arrive and park in any specified lots. This would allow for consumers to be more efficient.

2.4 Operating Environment (5 pts)

The operating environment of this software will be constrained to personal computers running any variant of the Windows operating system. This includes Windows 7/8/8.1/10/11. However, the system has not been properly tested on versions of Windows before Windows 10 and 11. The physical environment is not of any true concern, however it is recommended that the software is used in a safe location such as inside an academic building or simply away from moving traffic at any time.

2.5 Design and Implementation Constraints (5 pts)

Commands will be run in C++ and utilize the Simple and Fast Multimedia Library (SFML) to generate and display the graphical user interface. The System is self contained and will use batch files to compile the program. It can be run from the file directly without any further dependencies.

2.6 Assumptions and Dependencies (5 pts)

There are no additional assumptions.. The software is expected to perform offline.
Dependency on MongoDB database being accessible.

3. External Interface Requirements.

3.1 User Interfaces (5 pts)

N/A

3.2 Hardware Interfaces (2 pts)

N/A

3.3 Software Interfaces (2 pts)

MongoDB Database system, which contains VehicleIDs, locations, and Vehicle schedules. All of this information is provided by the database and object generation systems within the program.

4. System Features

4.1 Functional Requirements (40 pts)

Use Case ID: UC1
Use Case Name: Single Lot Simulation
Goal: Display and relay information of a specific parking lot.
Actors: Consumer
Pre:

- Program is running
- Program has displayed full map
- Program has generated vehicles

Post:

- Information about the specific parking lot

Main Success Scenario:

Step	Actor Action	Step	System Reaction
1	Simulation will be idle.	2	System will present parking lots for selection
3	User will select a parking lot	4	System will display the selected lot
		5	System will display lot variables/info
		6	System will present options to alter variables
7	User selects the option to alter variables. Variables include altered timestamps or closed lots not being available	8	System will run simulation with altered variables

Exceptions (Alternative Scenarios of Failure):

- User selected the incorrect parking lot
- User presses the back button to exit single-lot view

- Req. 1: The system shall display a full campus map
- Req. 2: The system shall present multiple parking lots to the user to select.
- Req. 3: The system shall display the selected lot
- Req. 4: The system shall display lot variables and vehicle information within the lot
- Req. 5: The system shall give the user the ability to alter the timestamp of the simulation
- Req. 6: The system shall run the simulation without crashing.

Use Case ID: UC2
Use Case Name: Whole Map Simulation
Goal: Display and relay information of the entire campus map
Actors: Consumer
Pre:

- Program is installed on the user's machine
- Program has generated vehicle and lot information
- Program is responding

Post:

- Capacity information for all present lots

Main Success Scenario:

Step	Actor Action	Step	System Reaction
1	User activates the system	2	System displays the whole map
		3	System will display preliminary object data
		5	System will present options to manipulate full map variables
		6	System will display preliminary parking lot data
7	User selects the option to alter variables	8	System will run simulation with altered variables

Exceptions (Alternative Scenarios of Failure):

- User selected the incorrect parking lot
- User presses the back button to return to day selection.

- Req. 7: The system shall display a full campus map
- Req. 8: The system shall present multiple parking lots
- Req. 9: The system shall display the capacity of a parking lot using a percentage
- Req. 10: The system shall present the user with options to alter the simulation variables
- Req. 12: The system shall run the simulation with user-altered variables

Use Case ID: UC3
Use Case Name: Lot Population Simulation
Goal: Populate and display the populations of various lots according to variables.
Actors: Consumer
Pre:

- Program is installed on the user's machine
- Program has access to vehicle and lot information
- Program is responding

Post:

- All present lots populated in accordance with the simulation and scheduling.

Main Success Scenario:

Step	Actor Action	Step	System Reaction
1	User activates the system	2	System displays the simulation start menu
		3	System will display preliminary object data
		5	System will present options to manipulate time variables
5	User selects a time and day constraint for the simulation	6	System will pull data and objects from the population according to constraints.
		7	System will run simulation within constraint and display proper results

Exceptions (Alternative Scenarios of Failure):

- Population in illegal state (improper time, improper locations, etc.)
- User presses the exit simulation button to terminate program program.

- Req. 13: The system shall present simulation controls
- Req. 14: The system shall use the population CSV to generate a population
- Req. 15: The system shall present the user with options to alter the simulation time
- Req. 16: The system shall run the simulation with user-altered variables

4.2 Non-functional Requirements

4.2.1 Performance Requirements (5 pts)

PE.1 The system shall not crash upon long term use
PE.2 The system shall not crash upon repeated simulations
PE.3 The system shall save the locations and information of vehicles to a file.
PE.4 The system shall read population information from an included CSV file.

4.2.2 Safety Requirements (5 pts)

N/A

4.2.3 Security Requirements (5 pts)

SE.1 Comply with Family Educational Rights and Privacy Act (FERPA)

5. Software Design

5.1 High Level Design (10 pts)

The system uses a mixture of layered and event-driven architecture, with each frame of the simulation being an upper level event that triggers the resulting events to take place within the frame. Within the frame, individual spots will either be empty or filled depending on the schedule of the population.

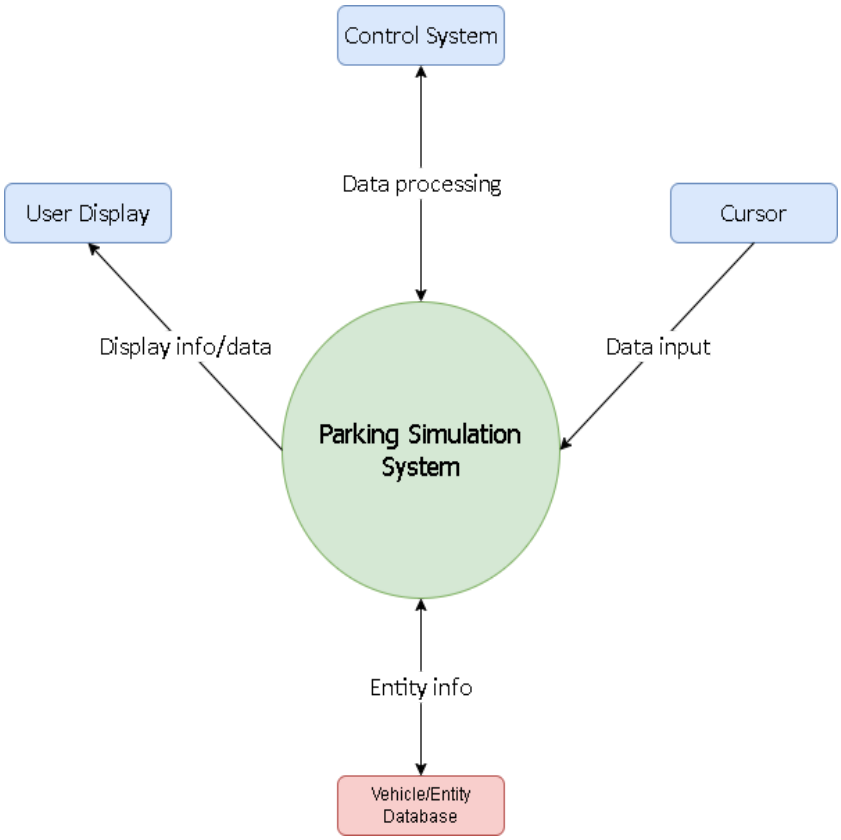


Figure 1. Context Diagram

As shown within the context diagram, the vast majority of the program's processes are within the core 'Parking Simulation System'. This core contains the generators, math models and object IDs that comprise the entirety of the system's backend. Layered on top of that backend is the User Display which works with the core in order to properly display the information to the user and properly scale that information for a variety of different display sizes. The control system handles the communications between the User Input or 'Cursor' and the core, while also providing updates to the User display in order to provide feedback.

5.2 Detailed Design (30 pts)

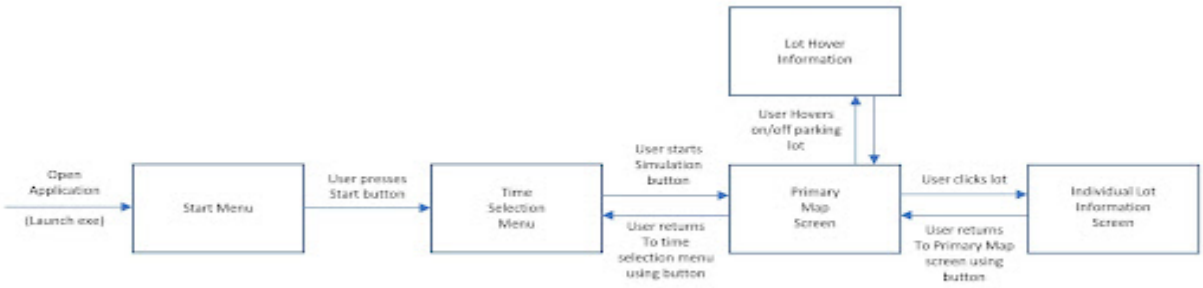


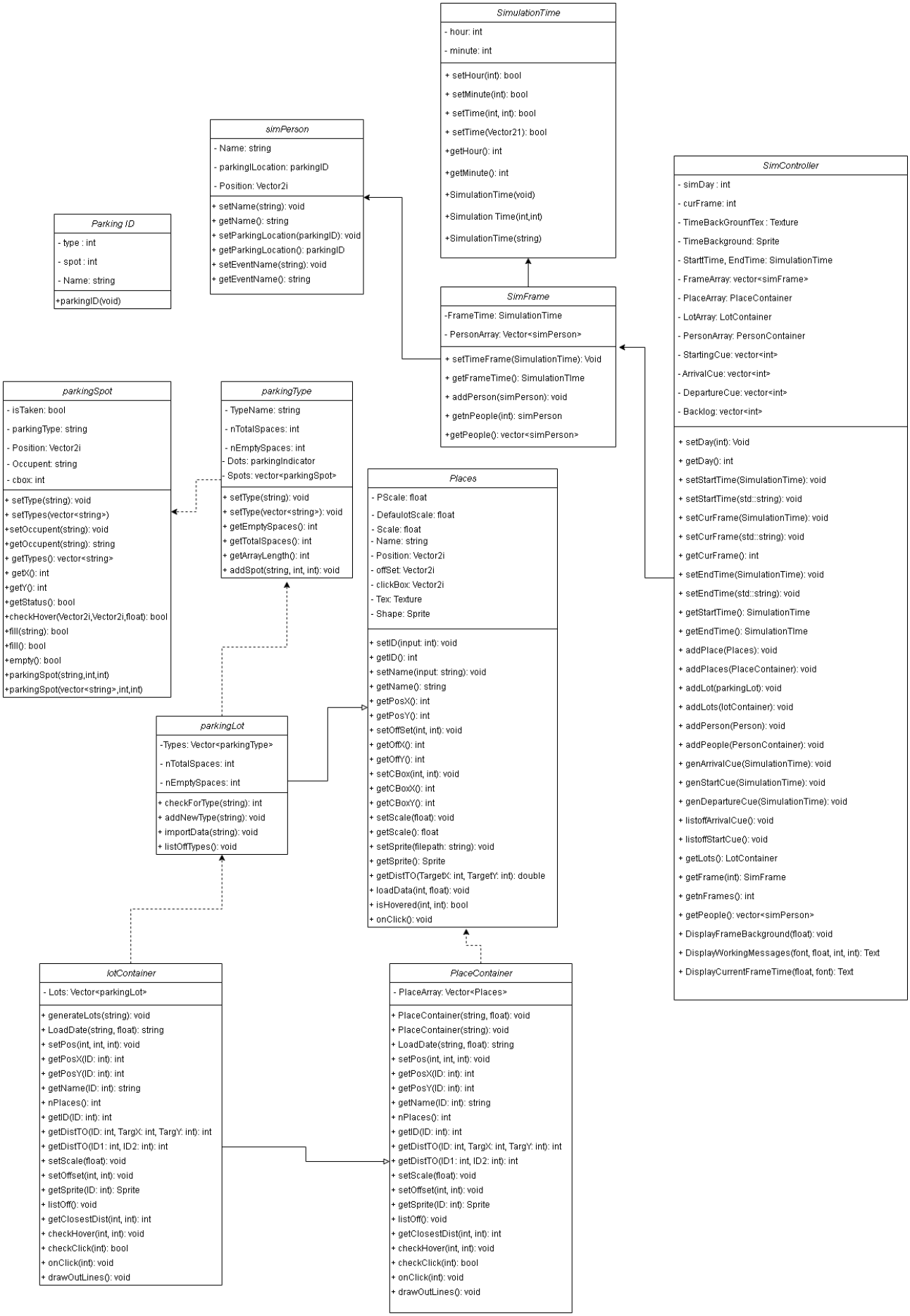
Figure 3. State Diagram

As shown by the state diagram above, the general design and functionality of the system is as follows: The system will launch into a start menu that displays the simulation variables, which are the time and date. The user can manipulate those variables as they see fit. Once the variables are selected, the user can press the "Start Simulation" button, which will lock in the variables and begin the simulation. The program then reads the population CSV and searches through it in order to display the correct vehicles and events as specified by the user on the previous page. This occurs at the same time as object and map generation. The moment the simulation finishes populating the map, the user will be brought to the main map screen where all 26 lots and buildings are present and outlined. The full map will provide preliminary capacity information on each lot in the form of percentages Beyond the main screen is the lot view screen, which will enlarge the images of the user's lot of choice and gives the user the ability to view the information of the specified parking lot: including the individuals and vehicles within them. The user will also have the ability to return to the main map view, as well as the simulation variable selection screen. If the user hits the "Exit Simulation" button, the system will terminate.

5.3 Detail Code

This is where you can submit your code. One option is to just enter a reference point to where the code is located (i.e., GIT).

GitHub Location: [JesseS15/512-Parking-Services (github.com)]


SimulationTime
- hour: int
- minute: int
+ setHour(int): bool
+ setMinute(int): bool
+ setTime(int, int): bool
+ setTime(Vector21): bool
+getHour(): int
+getMinute(): int
+SimulationTime(void)
+Simulation Time(int,int)
+SimulationTime(string)
simPerson
- Name: string
- parkingILocation: parkingID
- Position: Vector2i
+ setName(string): void
+ getName(): string
+ setParkingLocation(parkingID): void
+ getParkingLocation(): parkingID
+ setEventName(string): void
+ getEventName(): string
Parking ID
- type : int
- spot : int
- Name: string
+parkingID(void)
SimController
- simDay : int
- curFrame: int
- TimeBackGrounfTex : Texture
- TimeBackground: Sprite
- StarttTime, EndTime: SimulationTime
- FrameArray: vector<simFrame>
- PlaceArray: PlaceContainer
- LotArray: LotContainer
- PersonArray: PersonContainer
- StartingCue: vector<int>
- ArrivalCue: vector<int>
- DepartureCue: vector<int>
- Backlog: vector<int>
+ setDay(int): Void
+ getDay(): int
+ setStartTime(SimulationTime): void
+ setStartTime(std::string): void
+ setCurFrame(SimulationTime): void
+ setCurFrame(std::string): void
+ getCurFrame(): int
+ setEndTime(SimulationTime): void
+ setEndTime(std::string): void
+ getStartTime(): SimulationTime
+ getEndTime(): SimulationTIme
+ addPlace(Places): void
+ addPlaces(PlaceContainer): void
+ addLot(parkingLot): void
+ addLots(lotContainer): void
+ addPerson(Person): void
+ addPeople(PersonContainer): void
+ genArrivalCue(SimulationTime): void
+ genStartCue(SimulationTime): void
+ genDepartureCue(SimulationTime): void
+ listoffArrivalCue(): void
+ listoffStartCue(): void
+ getLots(): LotContainer
+ getFrame(int): SimFrame
+ getnFrames(): int
+ getPeople(): vector<simPerson>
+ DisplayFrameBackground(float): void
+ DisplayWorkingMessages(font, float, int, int): Text
+ DisplayCurrentFrameTime(float, font): Text
SimFrame
-FrameTime: SimulationTime
- PersonArray: Vector<simPerson>
+ setTimeFrame(SimulationTime): Void
+ getFrameTime(): SimulationTIme
+ addPerson(simPerson): void
+ getnPeople(int): simPerson
+getPeople(): vector<simPerson>
parkingSpot
- isTaken: bool
- parkingType: string
- Position: Vector2i
- Occupent: string
- cbox: int
+ setType(string): void
+ setTypes(vector<string>)
+setOccupent(string): void
+getOccupent(string): string
+ getTypes(): vector<string>
+ getX(): int
+getY(): int
+getStatus(): bool
+checkHover(Vector2i,Vector2i,float): bool
+fill(string): bool
+fill(): bool
+empty(): bool
+parkingSpot(string,int,int)
+parkingSpot(vector<string>,int,int)
parkingType
- TypeName: string
- nTotalSpaces: int
- nEmptySpaces: int
- Dots: parkingIndicator
- Spots: vector<parkingSpot>
+ setType(string): void
+ setType(vector<string>): void
+ getEmptySpaces(): int
+ getTotalSpaces(): int
+ getArrayLength(): int
+ addSpot(string, int, int): void
Places
- PScale: float
- DefaulotScale: float
- Scale: float
- Name: string
- Position: Vector2i
- offSet: Vector2i
- clickBox: Vector2i
- Tex: Texture
- Shape: Sprite
+ setID(input: int): void
+ getID(): int
+ setName(input: string): void
+ getName(): string
+ getPosX(): int
+ getPosY(): int
+ setOffSet(int, int): void
+ getOffX(): int
+ getOffY(): int
+ setCBox(int, int): void
+ getCBoxX(): int
+ getCBoxY(): int
+ setScale(float): void
+ getScale(): float
+ setSprite(filepath: string): void
+ getSprite(): Sprite
+ getDistTO(TargetX: int, TargetY: int): double
+ loadData(int, float): void
+ isHovered(int, int): bool
+ onClick(): void
parkingLot
-Types: Vector<parkingType>
- nTotalSpaces: int
- nEmptySpaces: int
+ checkForType(string): int
+ addNewType(string): void
+ importData(string): void
+ listOffTypes(): void
lotContainer
- Lots: Vector<parkingLot>
+ generateLots(string): void
+ LoadDate(string, float): string
+ setPos(int, int, int): void
+ getPosX(ID: int): int
+ getPosY(ID: int): int
+ getName(ID: int): string
+ nPlaces(): int
+ getID(ID: int): int
+ getDistTO(ID: int, TargX: int, TargY: int): int
+ getDistTO(ID1: int, ID2: int): int
+ setScale(float): void
+ setOffset(int, int): void
+ getSprite(ID: int): Sprite
+ listOff(): void
+ getClosestDist(int, int): int
+ checkHover(int, int): void
+ checkClick(int): bool
+ onClick(int): void
+ drawOutLines(): void
PlaceContainer
- PlaceArray: Vector<Places>
+ PlaceContainer(string, float): void
+ PlaceContainer(string): void
+ LoadDate(string, float): string
+ setPos(int, int, int): void
+ getPosX(ID: int): int
+ getPosY(ID: int): int
+ getName(ID: int): string
+ nPlaces(): int
+ getID(ID: int): int
+ getDistTO(ID: int, TargX: int, TargY: int): int
+ getDistTO(ID1: int, ID2: int): int
+ setScale(float): void
+ setOffset(int, int): void
+ getSprite(ID: int): Sprite
+ listOff(): void
+ getClosestDist(int, int): int
+ checkHover(int, int): void
+ checkClick(int): bool
+ onClick(int): void
+ drawOutLines(): void

The UML shown above details the various classes of the program and can be broken into multiple sections. The places and parking lots portion of the UML creates and manages the Parking Lots and Places of interest (POIs). Parking lots contain parking types, which contain parking spots. Allowing the program to keep all spots organized. The parking lot and place containers manage large numbers of their respective objects. The Simulation Time class allows for easy manipulation of time throughout the simulation. The Sim Events keep track of where people need to be and when. The simPerson keeps track of all of the attributes of the people in the simulation.

• 6. Test Cases and Results (25 pts)

Test ID: ATM_1	**Designed by: Sam T**	**Executed by (on date): Wade N. (4/13/22)**	
Purpose	Verify the integrity of the simulation, specifically its map data		
Dependencies	N/A		
Pass/ Fail Conditions	Successful program startup with map visible		
Pre-conditions:	Maps and lot data must be present within source file		
Test Data:	***Map = {ERAUMap, ERAUBaseMap}*** ***Scale = {N/A, Master_Scale}***		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Compile and Start program	ERAU Base Map should be visible	P
2.	Push 'Start Simulation' button	Frame should change to loading screen	P
3.		If successful, Scaled ERAU map should be visible	P
Comments	Test successful, the base map (an overhead view of the ERAU campus) appeared when the program was executed, and a properly scaled version of the map appeared when the simulation began.		

Test ID: ATM_2	**Designed by: Sam T.**	**Executed by (on date): Wade N. (4/13/22)**	
Purpose	Verify location of lot outlines		
Dependencies	***ATM_1***		
Pass/ Fail Conditions	Lot outlines must appear in the correct locations overlaid over the base map		
Pre-conditions:	• System must be up and running, • Map and lot data must be present within source file • Parking lot outlines must be present within source file		
Test Data:	***LoadData = from filepath*** ***SetSprite = from filepath***		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Compile and start program	Base Map visible	P

2.	Start simulation	Frame should change to loading screen	P
3.		Scaled map should be visible	P
		Outlines of parking lots overlaid on top of the scaled map in all locations	P
Comments	Outlines appeared as intended and required, unable to match cursor location with outline clickbox.		

Test ID: ATM_3	**Designed by: Sam T.**	**Executed by (on date): Wade N. (4/14/22)**	
Purpose	Ensure that lot outlines can be clicked on and generate a response		
Dependencies	List the identification numbers of all the test cases that must pass before this test case can be run, ***Example:*** • ***ATM_1*** • ***ATM_2***		
Pass/ Fail Conditions	Clicking on an outline clickbox should generate a response in the console that verifies the lot clicked		
Pre-conditions:	• System must be up and running, • Map and lot data must be present within source file • Parking lot outlines must be present within source file		
Test Data:	***CheckHover = true*** ***ListOff = lotNames***		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Start program	Basemap visible	P
2.	Start simulation	Loading screen visible	P
3.		Scaled map with lot outlines should appear	P
4.	User hovers cursor over lot	Clickbox responds and changes sprite	F
4	User selects lot	Lot name printed in console	P
Comments	The lots could be selected and the appropriate names appeared in the console (with one notable exception being Defender_ROTC) the sprite change upon hover did not occur as planned. Both issues have since been corrected.		

Test ID: ATM_4	**Designed by: Sam T.**	**Executed by (on date): Wade N. (4/14/22)**
Purpose	Ensure that expanded parking lot maps appear	
Dependencies	List the identification numbers of all the test cases that must pass before this test case can be run, ***Example:*** • ***ATM_1*** • ***ATM_2*** • ***ATM_3***	
Pass/ Fail Conditions	Clicking on an outline clickbox should load an enlarged image of the parking lot	
Pre-conditions:	• System must be up and running, • Map and lot data must be present within source file	

	• Parking lot outlines must be present within source file		
Test Data:	***CheckHover = true*** ***ListOff = lotNames*** ***LotMaps= filepath***		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Start program	Basemap visible	P
2.	Start simulation	Loading screen visible	P
3.		Scaled map with lot outlines should appear	P
4.	User hovers cursor over lot	Clickbox responds and changes sprite	P
5.	User selects lot	Expanded lot image appears	F
Comments	lot outline sprite corrected and functional, expanded lot images failed to appear due to filepath and scale issues. Both issues have since been corrected.		

Test ID: ATM_5	**Designed by: Wade N.**	**Executed by (on date): Wade N. (4/17/22)**	
Purpose	Ensure that parking spots appear in expanded lot views		
Dependencies	List the identification numbers of all the test cases that must pass before this test case can be run, ***Example:*** • ***ATM_1*** • ***ATM_2*** • ***ATM_3*** • ***ATM_4***		
Pass/ Fail Conditions	Parking indicators, empty or full must appear when a lot view is expanded		
Pre-conditions:	• System must be up and running, • Map and lot data must be present within source file • Parking lot outlines must be present within source file		
Test Data:	***CheckHover = true*** ***ListOff = lotNames*** ***SetSpriteSpot = filepath***		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Start program	Basemap visible	P
2.	Start simulation	Loading screen visible	P
3.		Scaled map with lot outlines should appear	P
4.	User hovers cursor over lot	Clickbox responds and changes sprite	P
5.	User selects lot	Expanded parking lot map appears	P
6.		Parking spot appear on the expanded map, properly placed, matching the image	P/F
Comments	Parking indicators appear, however not scaled properly with map. Pass/Fail		

Test ID: ATM_6	**Designed by: Sam T.**	**Executed by (on date): Wade N. (4/18/22)**	
Purpose	Ensure that lot outlines also possess a visible percentage		
Dependencies	List the identification numbers of all the test cases that must pass before this test case can be run, ***Example:*** • ***ATM_1*** • ***ATM_2***		
Pass/ Fail Conditions	Lot outlines must also display a percentage pertaining to the capacity of the lot		
Pre-conditions:	• System must be up and running, • Map and lot data must be present within source file • Parking lot outlines must be present within source file		
Test Data:	***CheckHover = true*** ***ListOff = lotNames*** ***getPercentFulle()***		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Start program	Basemap visible	P
2.	Start simulation	Loading screen visible	P
3.		Scaled map with lot outlines should appear	P
4.		Outline must also print percentage correctly	F
Comments	Percentage string appeared, however it was an incorrect percentage, Defender_ROTC had a capacity of -2,000,000,000% at some point, which was wildly wrong.		

Test ID: ATM_7	**Designed by: Wade N.**	**Executed by (on date): Wade N. (4/20/22)**	
Purpose	Ensure that the simulation controls are functional and recordable,		
Dependencies	• ***ATM_1*** • ***ATM_2***		
Pass/ Fail Conditions	On-screen controls must match with responses		
Pre-conditions:	• System must be up and running,		
Test Data:	***setFrameTime*** ***getFrameTime*** ***simDay = {m,t,w,th,f,s,s}***		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Start program	Basemap visible	P
2.		Simulation controls visible and clickable	P
3.	User hovers over variable	Variable GUI element glows cyan	P
4.	User selects variable	Variable GUI element glows blue	P

Comments	GUI elements responded to inputs from the user, and displayed as such within the console.

Test ID: ATM_8	**Designed by: Wade N.**	**Executed by (on date): Wade N. (4/20/22)**	
Purpose	Ensure that GUI slider positioning matches up with intended controls		
Dependencies	• ***ATM_1*** • ***ATM_2***		
Pass/ Fail Conditions	Lot outlines must also display a percentage pertaining to the capacity of the lot		
Pre-conditions:	• System must be up and running,		
Test Data:	***simStartTime*** ***simEndTime*** ***simDay = {m,t,w,th,f,s,s}***		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**
1.	Start program	Basemap visible	P
2.		Simulation controls visible and clickable	P
3.	User clicks on variable	Variable GUI element glows blue	P
4.	User drags slider	Selected variable GUI element changes to reflect slider movement	P
Comments	GUI and simulation responded to slider movement as expected, however, we encountered an un-repeatable event where pulling the slider too far caused the ‘day’ GUI element to print out the word ‘RUN’ rather than crashing with a pointer error. Team sufficiently spooked.		

Test ID: ATM_9	**Designed by: Wade N.**	**Executed by (on date): Wade N. (4/26/22)**	
Purpose	Ensure that the simulation can function under heavy load		
Dependencies	• ***ATM_1*** • ***ATM_2*** • ***ATM_3*** • ***ATM_4*** • ***ATM_5*** • ***ATM_6*** • ***ATM_7*** • ***ATM_8***		
Pass/ Fail Conditions	Simulation must function with a realistic campus population on any given day (~5000 Students/Faculty)		
Pre-conditions:	• System must be up and running, • Population CSV must be present within source folder		
Test Data:	***Population.loadData(filepath)*** ***simulationTime*** ***simDay = {m,t,w,th,f,s,s}***		
Steps	**Steps to Carry out the test**	**Expected behavior to be observed**	**P/ F**

1.	Start program	Basemap visible	P
2.		Simulation controls visible and clickable	P
3.	User selects simulation time and date	Variables updated awaiting simulation start	P
4.	User presses the “Start Simulation” button	Loading screen appears	P
		Parking lots and parking spots populated as the simulations loads	P
		Scaled map with outlines appear	P
	User drags time slider	Percentages align with lot capacity	P
	User clicks a parking lot	Expanded view with parking spots appears	P
	User drags time slider	Parking spots are filled and emptied as the frames progress until end of simulation	P
Comments	Simulation ran as expected and as hoped. However, start-up time could be measured in hours on smaller machines such as a Surface Go. It was learned that newer generation CPUs with more cores and higher thread counts reduced the generation time from hours to minutes.		

o 6.1 Traceability Matrix (10 pts)

Traceability matrix is used to keep track of what requirements have been developed, and tested. For each requirement, there will be one or more test cases.

Requirement	Test case	Results
Req.1	ATM-1	Pass
Req.8	ATM-2	Pass
Req.2	ATM-3	Pass
Req.3	ATM-4	Fail
Req.4	ATM-5	Pass/Fail
Req.9	ATM-6	Fail
Req.10	ATM-7	Pass
Req.5	ATM-8	Pass?
Req.6	ATM-9	Pass