# Cretaceous Gardens Controller (CGC)

*Software Architecture Design*

*SAD Version 3.0*

Team #1
26th November 2019

***CS 460 Software Engineering***

## 1. Introduction

This section provides an introduction to the Software Architecture Design document, helping developers and project managers understand the tasks required to achieve the goal of a successful park controller.

The CGC will be used as a software system that will monitor and control all interfaces of the island. The CGC will have control over the ticketing system, the authentication system, self-driving cars, and the security system. All of these interfaces have their own protocols that include normal operation and emergency mode.

The main idea of the architecture is to provide modularity and efficiency for later updating of the overall system.

This document will showcase the software architecture design that will be used for the implementation phase of the project. In section 2, it will showcase the design overview that will show the overall design and scope of the project. Section 3 will go over the component specifications and will go over function calls. Section 4 will showcase the sample use cases that helped shape the logic design. Section 5 showcases the design constraints and will briefly go over the code constraints that we will be using. Section 6 will go over the implementation.

# 2. Design Overview

*This section summarizes the design for the CGC on a technical level, providing an understanding of the way in which the system should be implemented.*

## 2.1 Design Approach

The CGC is comprised of four main components: the controller, the security system, the vehicles and the automated station. These components work together to allow visitors to experience the park in an automated and efficient manner. The controller communicates with the components to determine the overall state of the park and triggers emergency mode for all resources. All of the other components will manage their own behavior and send its status to the controller.

Each of the components that communicate with the controller manage their own internal interfaces. The security system manages several components of its own, the tranquilizer, the electric fence voltage monitor, infrared cameras and speakers. The vehicles manage a set of sensors and autonomously moves between the north and south ends of the island. The automated station manages a camera and a touch screen to allow visitors to get into the park.

## 2.2 Design diagram including all external interfaces

The overall architecture of the system is summarized in  figure 2.2.1. The system consists of numerous separate processes that communicate to control the park. The controller is the unifying component of the system, providing a summary of the states of the components.

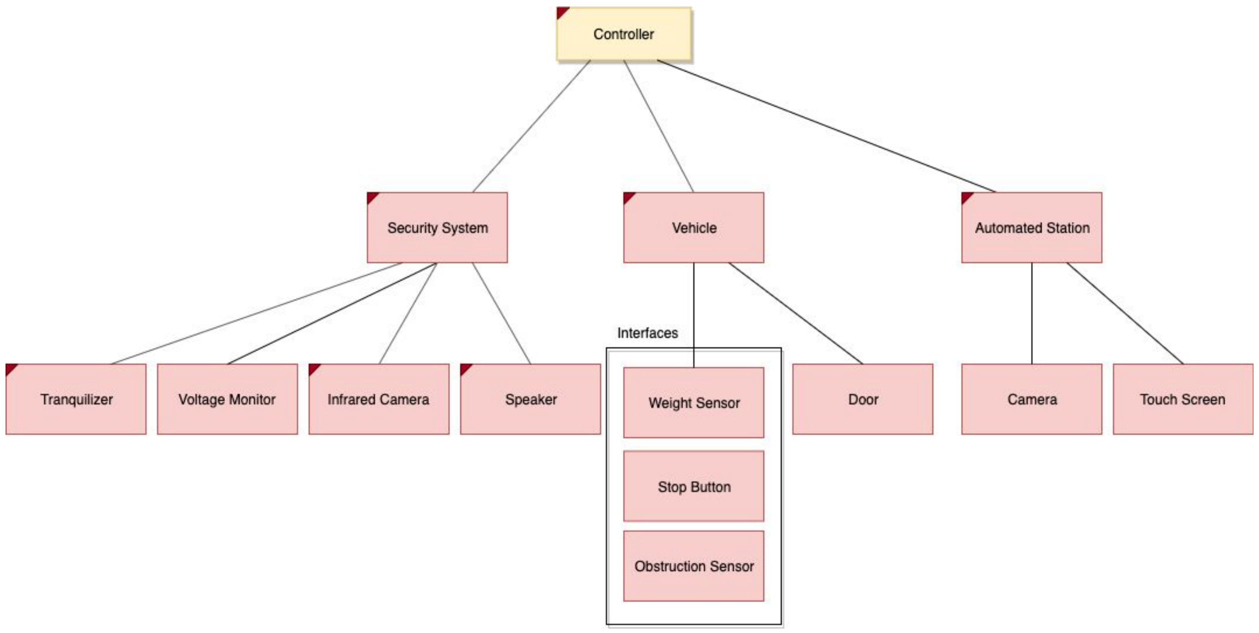

Figure 2.2.1: Overall communication layout of the CGC

# 3. Component Specification

## 3.1 Classes

The following are the classes that make up this software project.

### 3.1.1 Cretaceous Gardens Controller (CGC)

| Method | Caller |
|---|---|
| handleEvent(Object updateObj) -> void | Resource (Vehicle, Security System, Automated Teller, etc) |
| register(Object resource) -> boolean | Called by a resource to register |
| setEmergency(boolean emergency) -> void | Triggered by an emergency event or personnel |
| isEmergency()->boolean | Checks for emergency status. |

| | |
|---|---|
| registerAlert(Resource resource) -> void | Trigger when an event marks a resource as in an abnormal state |
| registerGuest(Guest guest) -> void | Register guests |
| getVehicles() -> List<Vehicle> | Get a list of Vehicles |
| getGuests() -> List<Guest> | Get a list of Guests |
| getStations() -> List<Stations> | Get a list of Stations |
| getSecuritySystem() -> SecuritySystem | Get Security System |
| registerZone() -> void | Registers a zone. |
| registerResource(Resource resource) | Registers a resource. |
| getZoneList()-> List<Zone> | Get Zone |

### 3.1.2 Automated Station

| Method | Caller |
|---|---|
| sendStatus() -> boolean | Triggered by internal event loop |
| takePhoto() -> void | Triggered by user input |
| validWaiver(Object waiver) -> boolean | Triggered by user completion of waiver |
| startTransaction() -> String | Triggered when payment is being processed |
| cancelTransaction(String transactionID) -> void | Triggered if transaction is failed/canceled |
| completeTransaction(String transactionID) -> void | Triggered to complete payment |
| registerVisitor(Guest guest) -> UUID | Retrieves UUID from CGC |
| setEmergency(boolean emergency) -> void | Triggered by message from CGC |
| depositToken(UUID userID) -> void | Triggered when the transaction is complete |
| registerZone(Zone zone) -> | |

### 3.1.3 Voltage Monitoring Sensor

| Method | Caller |
|---|---|
| getVoltage() -> float | Triggered by CGC |
| disable() -> void | sets voltage to 0.0 |
| enable() -> void | Sets voltage to 2.5 |

### 3.1.4 Infrared Camera

| Method | Caller |
|---|---|
| isAliceVisible() -> boolean | Triggered by CGC |

### 3.1.5 Vehicle

| Method | Caller |
|---|---|
| sendStatus() -> boolean | Triggered by internal event loop |
| stop() -> void | Triggered by button internal to vehicle, an obstacle is in front of the vehicle or when the vehicle arrives at the destination |
| verifyEntryRFID(UUID) -> bool | Triggered when card tapped on RFID reader |
| registerSeatRFID(UUID) -> bool | Visitor uses to take a seat |
| setEmergency(boolean emergency) -> void | Triggered by message from CGC |

| | |
|---|---|
| isOverCapacityDetected -> boolean | Trigger when the timer expires |
| isObstructionDetected() -> boolean | Triggered when the timer expires |
| move() -> void | Triggered when the timer expires |
| openDoor() -> void | Triggered when the ticket is valid |
| closeDoor() -> void | Triggered after passenger enters |
| playAudio(String audioPath)->void | Triggered by being in emergency mode |
| increaseCapacity() -> void | Increases currentCapacity |
| validate(Guest, List<UUID>) -> Boolean | Validates guests UUID. |
| checkCapacity() -> void | Checks vehicle capacity |
| checkMoving() -> void | Checks vehicle moving |
| Move(Double, Double) -> void | Moves Vehicle |
| getLocation -> Point | Gets Location |
| getShape -> Shape | Gets Shape |
| isMoving -> Boolean | Checks moving Vehicle |
| setMoving() -> void | Sets moving Vehicle |
| addToVehicle(Guest) -> void | Adds guests to vehicle |
| removeFromVehicle(Guest) -> void | Removes guests from vehicle |
| rotateVehicle(Double) -> void | Moves vehicle in GUI |
| getText() -> Text | Gets text. |
| playAudio(String)-> void | Plays audio |

### 3.1.6 Security System

| Method | Caller |
|---|---|
| sendStatus() -> bool | Triggered by internal event loop |
| setEmergency(boolean emergency) -> void | Triggered by message from CGC |
| playAudio(bool) -> void | Triggered by emergency |
| triggerTranquilizer() -> void | Triggered by emergency |

### 3.1.7 Guest

| | |
|---|---|
| getUUID() -> UUID | Get UUID |
| move() -> void | Moves guests |
| getLocations() -> Point | Gets locations of Guests |
| getIntersection(Vehicle) -> Boolean | Gets Bounds of guests |
| getShape() -> Shape | Gets circle |
| isInVehicle() -> boolean | Checks guests in vehicle |
| setInVehicle(Boolean) -> void | Sets guests in vehicle |
| setInvisible() -> boolean | hides guests |
| setVisible() -> void | Sets guests as Visible |

### 3.1.8 Resource

| | |
|---|---|
| sendStatus() -> boolean | Sends Status |
| setEmergency(boolean) -> void | Sets Emergency |

### 3.1.9 AppUpdate

| | |
|---|---|
| AppUpdate(boolean) | Updates app |

### 3.1.10 Zone

| | |
|---|---|
| DefaultZone -> Enum | Directions |
| getLocation() -> Point | Gets location |
| getHeight() -> int | Gets Height |
| getWidth(0 -> int | Gets Width |
| getRandomPoint() -> Point | Makes random Point |
| getShape() -> Shape | Gets Shape |

# 4. Sample Use Case

*This section covers examples of use cases that the system was designed with in mind. These cover what is seen as the most common, or important behavior of the overall system.*

## 4.1 Evacuate Protocol

*The Evacuation Protocol is a function used to start the emergency response throughout the park and get the guests to the entrance of the park in the most efficient and safest way possible.*

**Use Case:** evacuateProtocol

**Primary Actor:** Guests

**Goal in Context:** To evacuate the guests in the safest and most efficient way possible.

**Preconditions:** System must not be in emergency mode.

**Trigger:** The electric fence has become compromised/The T-Rex has escaped. Thus triggering an Alert signal to the CGC

**Scenario:**

1. The sensors in the fence will detect power shortages due to breakages or cuts in the fence.
2. If the cameras cannot find Alice within the enclosure the security alarm will send an alert signal to the Speakers, Autonomous Vehicles, and Automated Stations.
3. This will cause the guests to get into the nearest Autonomous vehicle so that they can get to safety.
4. The autonomous vehicles will drop the guests off at the entrance of the park and then go back and forth between the designated area and entrance.

**Exceptions:** If the Alarm is already on HIGH the alarm cannot be tripped again.

**Priority:** Urgent

**When Available:** Always

**Frequency of Use:** Hopefully never

**Channel to Primary Actor:** An Audio and Visual interface

**Secondary Actors:** The CGC control station: to determine when and where autonomous vehicles should leave and go.

**Channels to Secondary Actors:**

Speakers:

Autonomous Vehicles:

Automated Stations:

**Open Issues:**

1. How do we handle tranquilizer failure?

2. How are we keeping the guests safe if tranquilizers fails?

3. Unsure of the designated areas and how that function will work.

## 4.2 Autonomous Vehicle

*The Autonomous Vehicle is the transportation device used within the park to get our guests from the entrance of the park to the other side of the park to see Alice.*

**Use Case:** autonomousVehicleProtocol

**Primary Actor:** Autonomous Vehicles(AV)

**Goal in Context:** The autonomous vehicles will take guests from one side of the park to the other.

**Preconditions:** none.

**Trigger:** guests can find a ride near a designated area of interest.

**Scenario 1:**

1. Guests arrive at the park and after purchasing a token they need a ride to go see Alice.
2. After waiting in the designated area for a ride, the guests can use the tokens which has a unique RFID to get into one of the seats within the AV.
3. After a certain amount of time has passed the Vehicle with go to its next designated area.

**Scenario 2(Emergency):**

1. Guests will wait at the designated area for a ride.
2. The vehicles will not take tokens in an emergency response but will still stick to its capacity limit for safety standards.
3. After a certain amount of time has passed the Vehicle with go to the entrance of the park.

**Exceptions:**

1. If a vehicle has reached its capacity it will not move until other guests have left the vehicle.

2. AV will not let guests into one of its seats without a token.

**Priority:** Medium

**When Available:** Always

**Frequency of Use:** Hopefully never

**Channel to Primary Actor:** An Audio and Visual interface

**Secondary Actors:** The CGC control station: to determine when and where autonomous vehicles should leave and go.

**Channels to Secondary Actors:**

Speakers:

Guests:

**Open Issues:** unsure of the designated areas and how that function will work.

### 4.3 Security System

*The Security System is used to communicate any risks to the CGC.*

**Use Case:** securityProtocol

**Primary Actor:** CGC system

**Goal in Context:** The security system protocol will inform the CGC of an alert or emergency.

**Preconditions:** Evacuate Protocol must not be on.

**Trigger:** Fence / Cameras

**Scenario:**

1. If the fence loses power for a few seconds a risk alarm will be sent to the security system to let them know.
2. If the camera cannot track Alice a risk alarm will be sent to the security system to let them know.

**Exceptions:** If the emergency alarm is already set to on there is no reason for a risk alarm to be sent.

**Priority:** High

**When Available:** All times

**Frequency of Use:** whenever necessary

**Channel to Primary Actor:** The CGC control station: to determine when and what alarm needs to be sent.

**Secondary Actors:**

Camera:

Fence:

**Channels to Secondary Actors:** sensors / vision

**Open Issues:** none.

## 4.4 Automated Station

*The Automated station gives out a token to the guests of the park.*

**Use Case:** automatedStationProtocol

**Primary Actor:** Guests

**Goal in Context:** Guests will purchase their tokens from the automated stations to get the token which will help them get through the park.

**Preconditions:** Emergency protocol must not be active.

**Trigger:** guests interaction with some sort of currency transaction.

**Scenario:**

1. Guests will arrive at the park.
2. They will be guided towards the automated stations to confirm their transactions or at least make one.

3. To make a transaction the guests will see a familiar interface used to make a transaction to acquire a token which is used to access the park.
4. Once the transaction is made a token will be dispensed for the guests

**Exceptions:** Emergency Protocol is active.

**Priority:** High

**When Available:** always

**Frequency of Use:** whenever a customer needs to make a transaction.

**Channel to Primary Actor:** none

**Secondary Actors:**

Token:

**Channels to Secondary Actors:**

Completed transaction.

**Open Issues:** none.

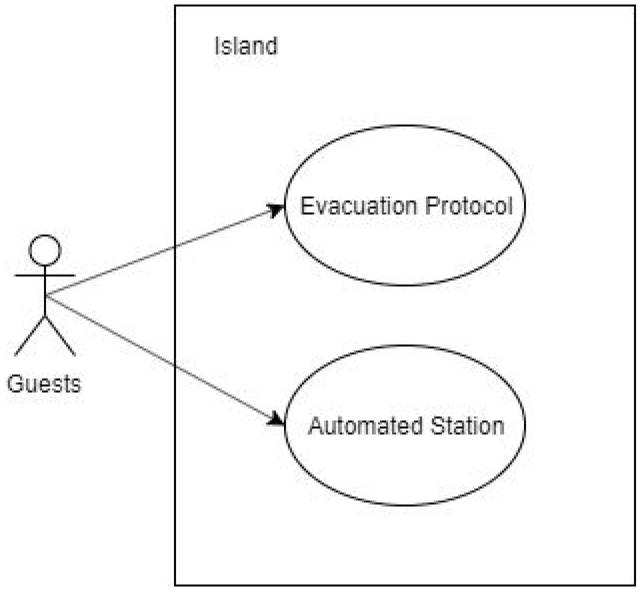

Figure 4.4.1: UML diagram of use cases for guests.

# 5. Design Constraints

*These section outlines the restrictions imposed on the system. These constraints include language to be used, implementation, and development standards.*

## 5.1 Programming Language

Java will be used to make the program. Java is an object orientated programming language which allows us to make the program modular. Java is also cross platform so it can be run on multiple operating systems. The Java has inbuilt support for GUIs (Javafx) so no external GUI libraries are not needed.

## 5.2 Tools

- **Git:** Used to track the different versions of the program. Also enables collaboration.

## 5.3 Implementation

**Modularity**: The system should be modular. The System must consist of different components that must interact correctly to make the entire system work properly. It is easy to maintain and upgrade the modular system in the future.

**Testability**: Each of the components of the system testable. The system should be modular to make it easier to test each of the components easily. If the testability of the system is high, finding faults in the system is easier.

**Coupling:** It indicates the relationship between modules or components. The system should be loosely coupled. Each component in the system must have have fewer dependency on other components. Classes that are tightly coupled are difficult to change and reuse. When something is change in one class, it should not affect the other classes.It makes it easier to maintain the code.

**Cohesion:** It indicates relationship of variables and methods inside the class. High cohesion indicates that a module has a specific task and only does that. High cohesion makes the program easier to maintain. When cohesion is high, the methods and variables of the class are interdependent. Classes must have a small number of instance variables. Each of the methods in the class should use one or more of those instance variables. The more variables a method uses the more cohesive the method is to its class.

## 5.4 Coding Standards

- Method names must be descriptive (usually verbs) and start with a lowercase letter.
- All class names must be descriptive (usually nouns) and start with an uppercase letter.
- All class and member variables (non-local variables) must be given descriptive names.
- At the top of every .java file, there must be a Javadoc comment describing what the class/interface is used for and how to use it
- At the top of every method, there must be a Javadoc comment block describing what the method does, its parameters, and its return value.
- Code blocks must indented to show the block structure with four spaces per level.
- No line must be more than 80 characters.

# 6. Implementation Plan

*The section summarizes the aspects that we plan to implement as part of the demonstration of the software.*

## 6.1 Simulation Overview

We plan to implement a simulation of the Cretaceous Gardens Theme Park using some of the built in Java Graphics libraries (JavaFX). The main simulation will be a display of the entire park, showing the different components and displaying their movement/states. The interface will provide a means to set different states of the park, allowing us to demonstrate how the system behaves under different circumstances.
The aspects of the park that won’t be implemented will be the details that don’t fit into the simulation.

1. Cameras will not provide video
2. Detection of Alice will be faked
3. RFID scanning when entering cars will be faked
4. Weight detection on cars will be a no-op
5. Alice will not escape and eat our visitors (unless we have time)
6. No users in the park will be malicious

## 6.2 Simulation Process

The simulation will be held within a jar file. You will be able to use different arguments to attain different results such as setting the amount of vehicles and passengers that will be in a given simulation. Most of the process will be run automatically with very little to no manual input after you set the arguments.

## 6.3 Timeline

This section breaks down the project into the three weeks that are given to complete the project. Each week the team will need to complete the defined goal to ensure a timely and successful simulation.

### 6.3.1 Week 1

Goal: Have a skeleton of the code necessary to start expanding the project
Tasks:

- Define common interfaces
    - Resources
    - 2D objects for displaying in the simulation
- Cover the API that is defined in Section 3
    - As least to the extent that we will implement
- Create a map and a path for vehicles to travel on
    - Some sort of GUI

### 6.3.2 Week 2

Goal: Have a GUI that displays components and most of logic completed
Tasks:

- Have a GUI for bare bones testing
- Complete the logic for the different components
    - To be tested using unit tests if GUI is not far enough along
- Have a complete GUI layout, even if not functioning

### 6.3.3 Week 3

Goal: Polish the project, ensure that the simulation is ready to be presented
Tasks:

- Cut extraneous parts of the simulation (if there are problems)
- Have a reliable part of the simulation to display to the class
- Prepare presentation
- Polish code/logic as much as possible

## 6.4 Early build of UI

This is an early build of the UI. It is missing a lot of features that we plan to implement but we can build on it.

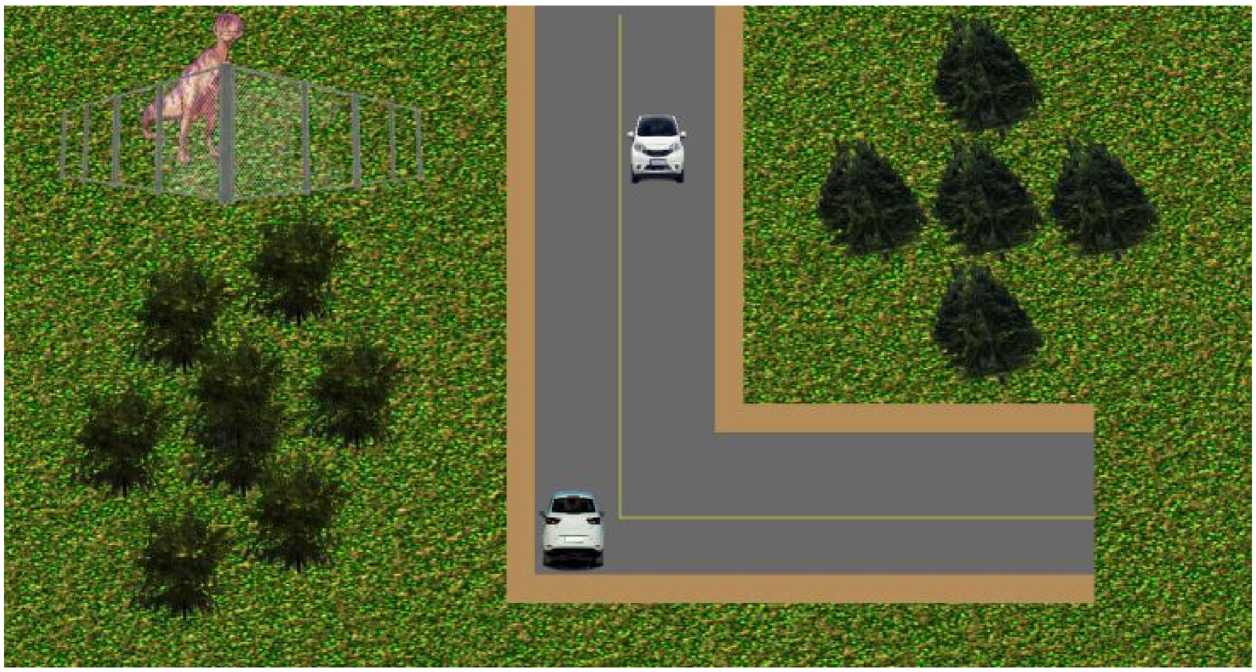

Figure 11: Showcases an early build of the UI that we have designed so far.

## 7. Definition of Terms

- **Alice** - The T-Rex and main attraction at the Island.
- **Automated Station -** Similar to a touchscreen ATM, this machine accepts cash and card payments and will print a token on valid payment.
- **RFID (Radio Frequency Identification)** - Used to identify visitors using a unique token
- **Tranquilizer Device** - a device implanted into the dinosaur that allows personnel to remotely tranquilize the dinosaur in emergencies.