The Giving Child

Cici Collier Kellyn Larson Mike McClary

Griffin Metz Vanessa Ramos Nick Zustak

CSCI 370
June 2016

Introduction

The Giving Child is a non-profit organization that puts an emphasis on developing mobile applications that empower children to help change the world. Their goal this year is to guide children towards being knowledgeable about animal endangerment. We were tasked with designing an application that raises awareness about this issue by allowing the user to create and maintain their own virtual animal wildlife sanctuary. The game requires the user to use an in-game currency to build enclosures for animals, interact with animals that come to the enclosures, and ensure that all of the animals maintain a certain happiness level. We were also required to provide information about different endangered species within the game to help educate the children who play the game. 90% of the game's purchasing price will go towards select animal protection agencies. This game is targeted at a younger audience, but is easily played and enjoyed by all ages.

Requirements

<u>Functional requirements:</u>

The app is supposed to be engaging for younger children. They should be able to interact with animals in a positive manner and see how taking care of animals makes them happy. Our client left this project very open ended, so our team came up with the idea to show animals in enclosures from a bird's eye view, and be able to click on them to start an interaction screen. Eventually, you can add new enclosures to your sanctuary to bring in more animals.

- Display the sanctuary
    - Include where the current enclosures are built, which animals are in the currently-built enclosures, and where there is available space to build other enclosures.
- Display available options on the sanctuary screen through buttons
    - New day option that can be chosen after at least one animal has been tended to and after all events have been taken care of. If these requirements have been met then a new day will begin. The animal's happiness will decrease and there is a chance that an event will occur that day.

    - Expand option that can be chosen once at least 15 stars have been collected from dealing with events. This will bring the user to choose the environment for a new enclosure, and then outline the shape and size of the new enclosure.
    - Menu option that will take the user to the main menu with the options to see the instructions and to restart the game.
- Use buttons to let the user specify details of their enclosure
    - The size and shape are specified by the user selecting squares in the sanctuary that will be enclosed together by a fence. The user will be able to choose the environment type. The environment type will determine the ground shown enclosed by the fence on the sanctuary screen and the background behind the animal that appears in that enclosure. Animals will appear in environments that fit their species type. For example, a lion will appear in the Savannah and not in the Mountains.
- Make an animal appear in new enclosures.
    - When an enclosure has been successfully made, an animal will appear in the sanctuary screen. Its species will be randomly selected from a selection based on which environment was chosen. The animal will also be given a random name and a gender.
- Transition from looking at sanctuary to individual animals.
    - When an animal in the sanctuary is clicked on, the screen will transition to being one with an environment background, a close-up of the animal, and buttons that will take the user to a facts-generating screen, take the user back to the sanctuary, and let the user interact with the animal.
- Use buttons to interact with the animal with animations illustrating each interaction type.
    - Three different interactions: play, feed, and wash. Each button will increase the animal's happiness a certain amount before beginning to decrease it. A warning dialog does pop up before the happiness starts to decrease. If the user continues to do a certain activity after the animal is tired, the happiness bar will start to lower. Which could lead to the animal dying.
- Have a way to learn about the animals
    - A facts screen on which the user can click through facts about the kind of animal currently being tended to. This screen can be accessed from the animal interaction screen. One fact will be displayed at a time until the user clicks Generate New Fact.
- Have daily random events
    - An event can occur after the user clicks the New Day button in the Sanctuary Screen. We implemented four different events: poacher, tornado, sickness and fire. These events appear randomly when a new day begins. The user must take

care of the event by clicking on it until it has disappeared. Once the user takes care of the event the user will receive a star, or 3 stars for taking out 3 fires.

- Give an option to go “back” to your previous screen
- Have continual saving that the user doesn’t need to worry about
- Give an option to reset the game and start over

Non-functional requirements:

- Code the program in Java
- Use Eclipse to program the game
- Use libGDX to make the program Android and iOS compliant
- Write to and read from text files to save the data
- Use GitHub for version control and easy access

System Architecture

The final UML diagram for the application code is shown in Figure 1 below. The main architecture of the game contains the game engine, the game screens, and helper classes. The game engine is in charge of creating and starting the application in various platforms. The screens create the different screens that the user will see and interact with. The helper classes provide backend functionality for the game such as loading texture files.

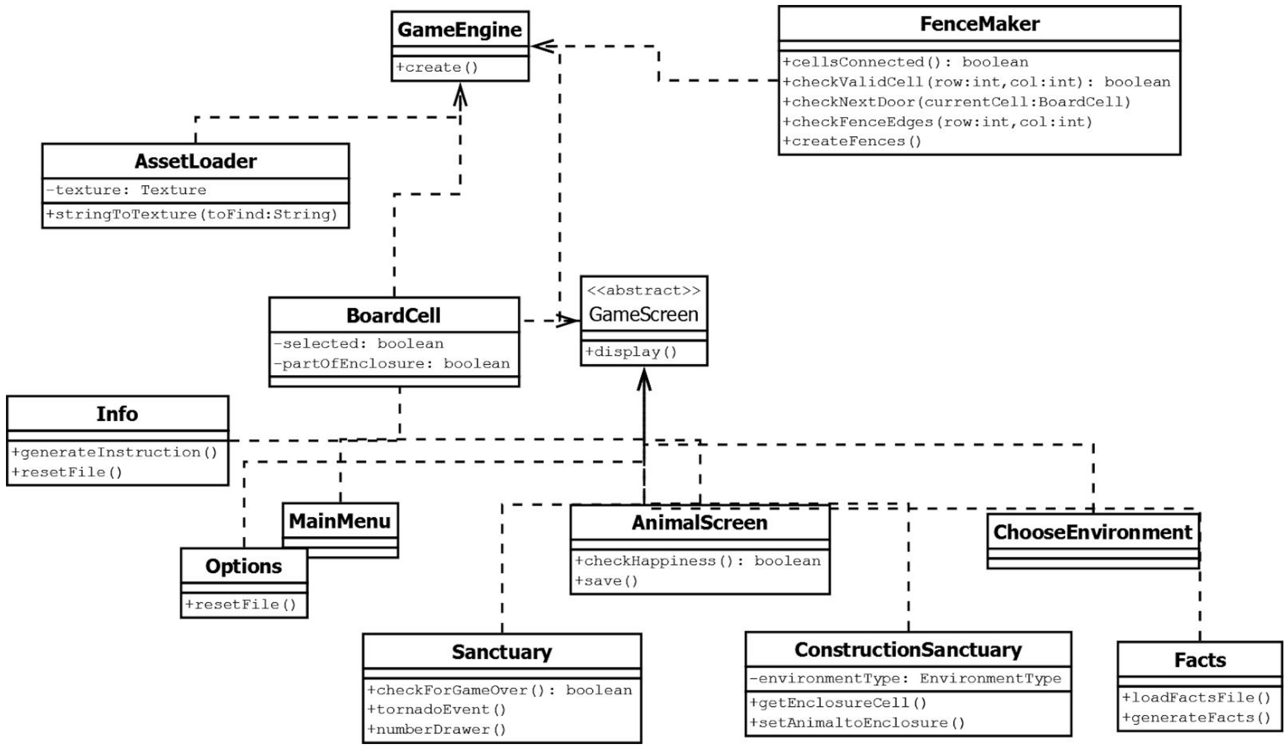

FIGURE 1. UML Diagram

- The AssetLoader class acts as our own library of things we need to use throughout the game. This includes everything from the animal graphics to the button graphics. We load the images and initialize animations that will be used throughout the game. The AssetLoader class helps us keep all of our attributes in one place.
- The BoardCell class contains the individual square cells of the sanctuary. Each cell holds a selected texture (for displaying on the sanctuary screen), an unselected texture (for when you click on it in the construction screen), and a boolean indicating whether or not the cell is a part of an enclosure (to determine whether or not you can click on it in the construction screen)
- The AnimalScreen is where the user can see a close-up of the animal they've chosen, choose to see facts about that animal, interact with the animal by feeding it, washing it, or playing with it, or go back to the sanctuary screen. This screen loads each animal

depending on the animal clicked in the SanctuaryScreen. The animal's information is loaded to this screen by using their unique name as a key.

- The SanctuaryScreen is where the entire sanctuary can be viewed in bird's eye view form. This includes showing where the current enclosures are, which environments they represent, and which animals are in them. The animals on this screen are clickable and will take you to their respective animal screens as explained above. There are next day, expand, and menu buttons at the bottom of the screen. When next day is chosen, a clock animation will play through. If an event were to occur, it would be displayed in this screen as an icon. These events will show up on the sanctuary and must be dealt with before the user can move on to a different day.
- The FactsScreen can be accessed through the animal screen, and is a screen with an option to generate a fact about the animal you're playing with. This button can be clicked as many times as the user would like to click it. The program reads from a list of facts, and randomly chooses one each time the button is pressed.
- The ChooseEnvironmentScreen is accessed from the sanctuary screen through the expand button. It is a transition screen where the user can choose an environment. Once the environment has been chosen, the user will be taken to the construction screen.
- The ConstructionSanctuaryScreen shows the current sanctuary, but here the user can click on available squares to make a new enclosure. The submit button on this screen will display a dialog if the enclosure is not valid. An enclosure is valid if it contains more than one square and all squares are connected. An animal will be created in this screen and the user will be notified that they have a new animal. Once the submission is complete, the user will be taken back to the sanctuary screen where they can see their new enclosure and animal.
- The FenceMaker class is a helper class used when the user creates a new enclosure. The class contains the algorithms that determine whether the user created a valid enclosure as well as a method to set the correct textures to the cells to make the fences appear on screen. This class is used within the construction sanctuary screen when the user submits the cells they want for a new enclosure.

Technical Design

FenceMaker Algorithms:

The FenceMaker implements a few interesting algorithms. The two main algorithms are cellsConnected() and createFences(). CellsConnected checks to make sure all of the selected cells are connected and make a valid enclosure (there are at least two squares and everything is connected vertically and horizontally). This function calls a recursive helper function called checkNextDoor. This function simply checks each cell above, below, and to either side of it to make sure it fits the criteria for being a selected cell and adds that cell to an arrayList of visited cells.

CreateFences() checks every cell to see if it's selected, and then checks above, below, and to either side of it to determine if there should be a fence. If above, below, or to either side is a selected cell, then there isn't a fence, if it's not selected or if the selected cell is on the edge of the grid, that side gets a fence. Each side of the cell is marked with a 0 (no fence) or a 1 (fence) reading from the top of the cell clockwise. Hence, a cell with a fence on the top and right side would be denoted as "1100". This, in addition to the previously selected cell environment, is then written to a file.

Display Algorithms:

In our sanctuary screen, we need to display the number of days that has passed. We do this with a function called numberDrawer. In this function, we take the integer number of days and mod it by 10 to get the last digit of the number (for the number 1234, the value we find would be 4). We then check the number of digits in the number, and set an offset factor accordingly, then draw that last digit.We then divide the number by 10 to truncate it (1234 becomes 123), and start the process over. The first digit (1) will always be written directly to the right of "Days", and the offset factor pushes the digits to the right accordingly.

Dynamic Resizing Dependent on Screen Resolution:

In creating the game, it quickly became apparent that hard-coding number values for button sizes, dialogs, grid sizes etc. causes problems when deploying to different platforms. To get around this, nearly every sizing parameter is based on the current screen width and height, which the libGDX API is able to retrieve at any point in time. By making all the placements of all the actors based on a fraction of the height or the width, everything shows up on the same place no matter what device you're playing the game on.

AssetLoader:

The AssetLoader allows all of the game graphics, animations, and other files to be stored in one place. The files can then be loaded once at the start of the game and stored as static variables. These variables are referenced throughout the code, so the assets are not being loaded every time they are needed. This structure helps prevents memory over usage by ensuring the graphics files are stored in memory only one time. Another benefit of the AssetLoader class is the overall performance improvement of the application. The application is graphics intensive so constantly loading textures would slow the game. Instead, the AssetLoader results in a few seconds of loading time at the start of the game in exchange for improved performance throughout the rest of the gameplay. The AssetLoader also contains the stringToTexture function which allows us to conveniently load the correct texture cell for displaying the fences and enclosures in the sanctuary. Lastly, the dispose function removes all textures from memory at the end of the app.

Changing between screens:

The ability to change between screens comes with pressing a button that creates a new screen. The flow between screens can be seen in the application storyboard seen below in Figure 2. The storyboard also shows the buttons that the user clicks to transition between screen. Clicklisteners are used to listen to user input, and a new instance of the appropriate screen is created and set so the user is transitioned to that screen. While we had the option to be able to show and hide screens, we figured it would be the least memory intensive and easiest to make new screens as we went along. This is because we were saving data to txt files and didn’t know how to manipulate them between multiple open screens.

Custom ClickListener:

We created our own Click Listener to help click on BoardCells in the construction sanctuary screen. The Click Listener, when called, finds where you clicked on the screen and translates that into an integer number that defines which BoardCell was clicked. Once the listener is called, any necessary code will be updated. Figure 3 shows a flowchart for the Click Listener.

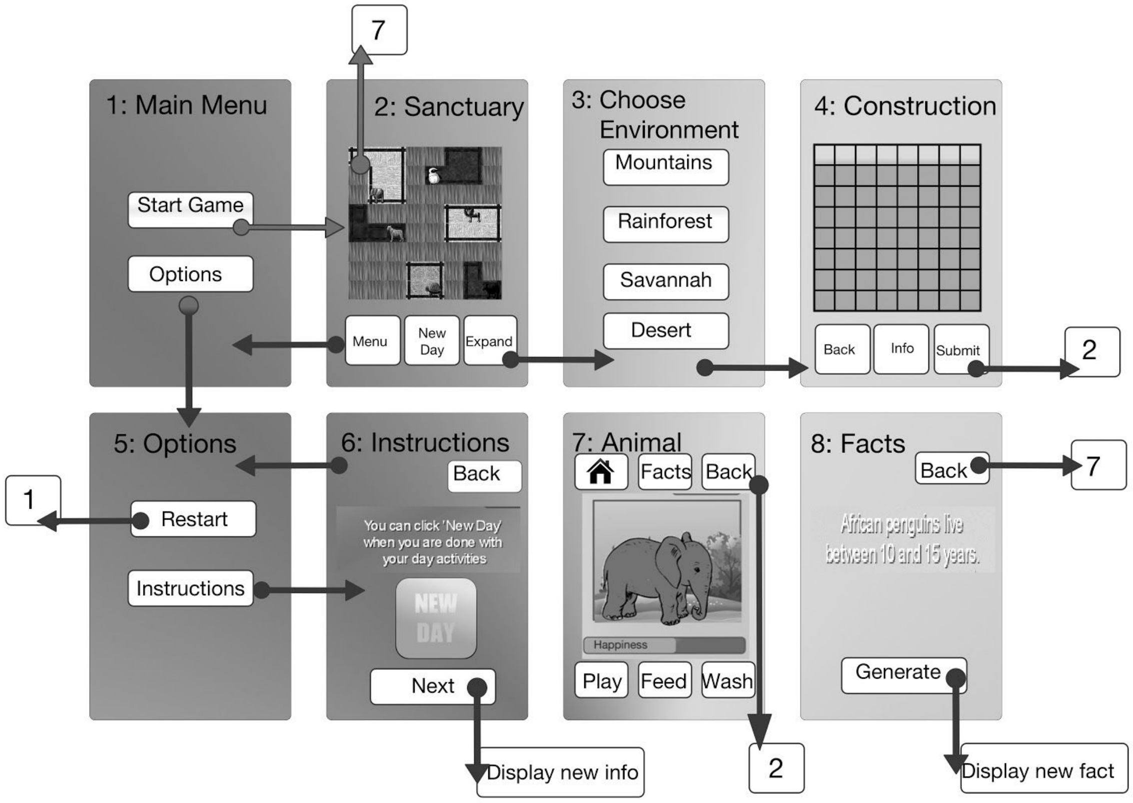

FIGURE 2. Storyboard

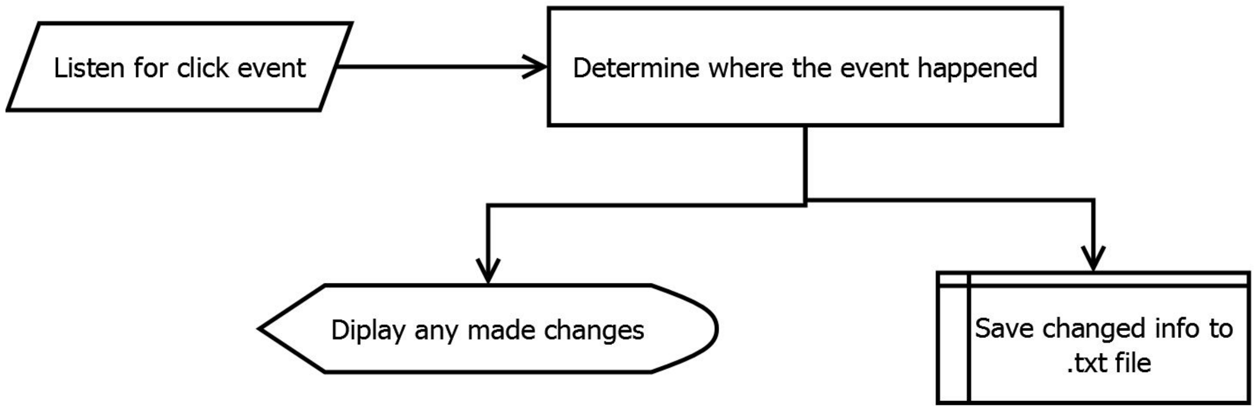
Listen for click event
Determine where the event happened
Diplay any made changes
Save changed info to .txt file

Figure 3. Click Listener

Decisions

- Instead of using Unity, a cross platform game engine, we decided to use LibGDX, a free and open source library. Our client required that our app runs on iOS and Android and LibGDX allowed us to write one core codebase in Java, and then package/deploy it to Desktop, Android, Web, and iOS. Unity is also licensed and required us to code in C#. Unity specializes in 3D rendering, whereas we only needed 2D graphics for which LibGDX is perfect. LibGDX allowed us to make more software design choices and write more of our core game code, which we felt was an important part of this project.
- We chose to use Eclipse for our IDE because it was the IDE we all had experience writing Java in. It also has all the plugins needed to use LibGDX and run iOS and Android simulations. We also considered coding in Android Studio or Visual Studio, but we determined that Eclipse would be easiest to export our project to multiple platforms such as iOS and Android.
- We decided to store data in local files private to the app rather than using a database. Databases are not a built in feature for LibGDX so adding it in would be complicated- especially across both Android and iOS. File I/O is both much simpler to implement and covers what we need it to for the purposes of our project.
- We implemented an AssetLoader to manage and load all our assets at once instead of loading each asset individually as we use it throughout our code. This allows us to have a single place to store all our assets and make our game memory efficient by only loading an asset once.

Results

Features we did not have time to implement:

- Sounds
- Ability to switch between platforms with the same account/data
- Breeding animals
- Fence construction freedom
- Time limit for fixing the fences
- Gathering resources to build the fences
- Water supply
- Planting grass
- Building structures
- Animal tracks
- Collars with bells
- River baths and forest walks
- Skinny animals
- Growing brush
- More than one animal in an enclosure if there's space

Performance testing results:

This project has been tested on Desktop, iOS, and Android. The game runs on all platforms, but looks best on the mobile platforms as those are the platforms the game was designed for. Apart of a few seconds of loading at the beginning, the app runs smoothly with minimal lag or delays.

Summary of testing:

Because our project is a game driven by user input, most of our testing could be done by simply running the app on the various platforms and manually using the UI. For example, if we needed to test our ClickListener we would need to run the app on the Desktop and manually play the game to check that the clicks were working properly. Sometimes this was very frustrating, since for some tests we would need to run through the game up to a certain point. This was time consuming, and sometimes only resulted in a small piece of information. In addition the saving of the game is done through text files, so to test this we could just run the game and view the text files. However, for some of the backend functionality, such as our fence-making algorithm, we did implement JUnit tests to ensure that everything was working as intended.

Lessons learned:

- LibGDX provides a simple way to deploy a simple code base to different platforms including desktop, Android, and iOS. This made it relatively simple to develop one app for multiple platforms.
- It's important to set realistic goals for your project and not get caught up trying to add unnecessary features early on in the process. Fortunately we got this advice from both our advisor, Dr. Paone, and Walter, one of our clients. Following this advice we created a priority list of all the features we were thinking about adding, ranking them as "must have," "nice to have," and "low priority." This kept our development focused and made for a relatively stress-free work flow.
- Good communication with the client keeps the process going smoothly.
- Your team may think you're all on the same page, but when implementation of the design starts, it could turn out that everyone has a different idea and you need to take a step back to figure out what's actually going to happen.
- Sometimes it just takes a fresh set of eyes to see where the error is in your code, whether or not those eyes belong to someone who is better at coding than you or not.
- Showing up on time and not leaving early are good ways to show your teammates that you're there to help get things done and be a part of the team.
- Even though you may think you're good at giving presentations, it's still a good idea to practice a few times before the actual thing because nerves will make you forget things, but at least if you've practiced you've probably recovered from the same part before.
- If you buckle down and focus, a team of six can get a lot done in only a few hours.
- Graphics take longer than you'd think, so make sure you start small and work up from there.
- There will always be little bugs hidden in your code when you think you're basically done, so make sure you save time before your project is due to fix small errors.
- Everyone has a different style and skillset, so come in to a team prepared to compromise with, listen to, and learn from your teammates.

Ideas for extension:

- Be able to add more animals to an enclosure to be able to have multiple animals together.
    - Rather than only having one animal per enclosure regardless of the enclosure size, there could be one animal per a certain number of cells
- Extend the size of the grid depending on the number of days that have passed.
    - Eventually, the user will fill up his or her sanctuary.
- Have the animals move around their enclosures on the sanctuary screen
    - This would look more enticing and be slightly more interactive with the user
- Include more animations
    - Adding more animations, such as the animals breathing or the wind blowing, would make the game more enticing and more real
- Add option to build addition features, both for animals and environment
    - This would allow for more complex interactions with the animal and in the construction screen
- Include resources around the sanctuary to gather to be able to build and do more
    - Instead of being given everything, the player could have to gather sticks themselves in order to build the enclosures
- Incorporate sound into game events
    - For every action that happens, we can have a custom sound play
- Expand upon the happiness meter
    - Include a boredom, hunger and thirst level
- Add more animals and more environments
    - The more environments, the more choices the user has in designing their sanctuary. However, the environments take more time to make than they seem.
    - The more animals and environments, the more unique the user's sanctuary can be.
- Create an interactive storyline for the user to enjoy
    - Introduce a basic storyline to keep the user entertained. At certain points throughout the game, a new addition to the story could be added to keep the user interested.