Release Management Plan - G.T.H.C

Game Tenting Help Center (GTHC)

This document serves to describe how the team intends to manage the release of the application so that anyone can read and understand how our application will be distinguished from the current development version as well as any dependencies. This document serves to clarify packaging and delivery for our product.

Software Configuration Management (SCM)

Our team intends to ensure that the dynamically changing aspects of our products are clearly laid out. Any kind of change or modification to the application in the future will have to go through the following steps:

1. Change proposal
2. Identify components that are needed to be changed
3. Review and evaluate that proposal
4. Approve or Deny the change
5. Implement change
6. Test the change
7. Notify users after fully accepting the change

By going through these thorough steps for any kind of modification, we can ensure that the client will be satisfied when we need to roll through with any changes/modification. Ideally, the changes are implemented in the offseason of tenting so that taking down and putting the site backup is not a problem. However, if any immediate altering is necessary then we will implement as needed.

Technicalities:
Before making a release, we will need to have the features of a release broken down into tasks for each team member to work on. Each task will be worked on it's own branch, and a completed task will be assigned to a Pull Request. This will allow us to individually test each tasks and/or features before merging them and sending it into production. Moreover, using pull requests will allow us to setup Continuous Integration tools such as CircleCI or Travis to automate tests to make sure the new feature is functional on the higher level, and it is also the responsibility of other team members to review their peers' Pull Requests before pushing onto a higher branch. Reviewing Pull Requests require the reviewer to look both at code quality and functionality before

approving a merge. Each Pull Request has been set up to require team approval before pulling. Overall, the stages are the following:

Standalone branch for feature/task → Pull Request → CI → Team Approval → Staging Branch (Built and Deployed on staging) → Master Branch (Build and Deployed on production)

Build Management

At the moment, Heroku is our cloud platform of choice because it handles automated build process when deploying our projects. With Heroku's automated deployment, we can make changes to our staging branches to our GitHub repositories (which Heroku can gain direct access to), and automate the entire build process from the moment that a pull request is given access to merge onto the staging branches.

Packaging

As we are using the node js framework, all of our dependencies are managed by node's packaging manager (npm), and in both development and on our production build, we use yarn as our main dependency management tool, which has been proven to be a faster, more space efficient, and a better developer experience alternative to npm.

Releasing to Production

When building any application that will eventually go into development, the main goal is to always replicate the production environment in development. Typically, the best solution is to set up a container that has its environment setup in a uniform fashion. For example, a docker container running an instance of ubuntu that has the same installation process each time you run it up for any depencies you may need, such as npm, and react. Due to the complexity and general team experience, we have decided to manually maintain a uniform environment in development. Maintaining our environment will make deployment to production much easier, as we can be assured with what works and does not work in development will be the same in production. Before releasing, we follow the steps mentioned above in SCM, and we will use our cloud platform (most likely Heroku, but could potentially be GCP or AWS) to build, run, and host our application.

Defect Tracking

As mentioned before, we can use a Continuous Integration (CI) tool, preferably CI, that will run tests automatically on PR before merges are allowed to catch any defects that we test for. This is useful because it will stop us from pushing defective code.