Hello Davide,

Thanks for taking the time to check out this test and hopefully complete it. We'll try to use your time in the best way possible to evaluate the match between your experience and our company.

These exercises are here to evaluate your capability to assess our specific vision given the least amount of details from our side, and let us understand your working pipeline. We are generally more interested in the process itself than the final result, so please prioritize the specific requirements over the final output. Try to do your best in terms of polishing, code clarity, naming conventions and proper comments in the code - we care about these things.

For each assignment you have to deliver a Google Drive folder properly named which contains:

ROOT FOLDER (Project Name) - NAME SURNAME TEST - COMPLETION DATE

- **00_References**
    - Collect a list of videos of the features you have implemented
- **01_Build**
    - Win64 build of the project with a working executable, in a single .zip file

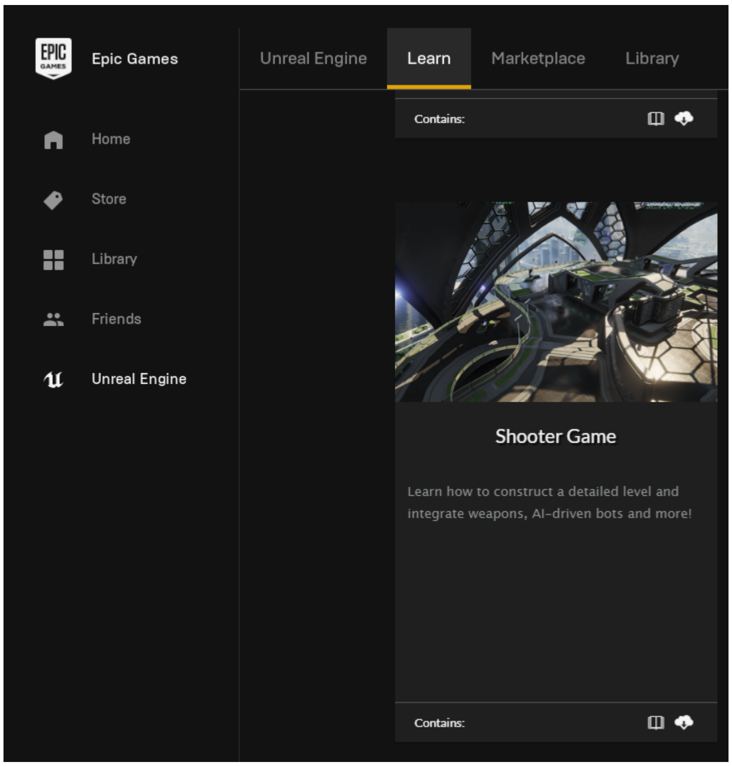
Once you are done, please share them with: vdidonato@34bigthings.com, mtrezza@34bigthings.com, gportelli@34bigthings.com, gferronato@34bigthings.com

Also, the project must be on a GIT repository that you will give us access to (keep in mind we'll be assessing your commit history, .gitignore and .gitattributes setup, as well as the code itself).

For the GIT repository, please add the following users: vdidonato@34bigthings.com, mtrezza@34bigthings.com, gportelli@34bigthings.com, telivyel@gmail.com

Shooter Game Ex #1

Applicants are required to implement some modifiers (**minimum three**) for the "Shooter Game" example project downloadable for free from the Epic Games Launcher -> Learn.



The three modifiers of choice need to be chosen from the following list:

- **Teleport:** the player is able to teleport 10 meters forward upon pressing the T key. Although this mechanic is very easy to implement in a single player game, we ask you to implement it for a multiplayer environment where the integration with the internal character replication system is not trivial. The right way to do this on a UE4 Character is by adding a new movement ability to the character movement component. Follow this documentation for your implementation: Advanced Topic: Adding New Movement Abilities to Character Movement using approach number 4. Make sure to stress this mechanic on multiplayer by simulating a large packet lag (see here for documentation). Use a lag simulation of 500ms with 30% lag variance.
- **Jetpack**: players have an energy pool they can use to fly simulating a jetpack. The energy gets recharged when not flying and it's properly displayed on the HUD.

- **Wall jump**: the player is able to perform a lateral jump while flying very close to a wall. This jump will push the player to the opposite direction of the wall, and a bit higher than before.
- **Freezing gun**: when hitting an opponent with this weapon it gets freezed and cannot move or shoot for a specific amount of time. Proper graphic effect is a plus.
- **Shrink gun**: when hitting an opponent with this weapon it gets tiny (about 20cm tall) for a specific amount of time. If during this time another player stomps on him, then he dies. The shrinking, and eventually subsequent rescaling, must be interpolated. The amount of remaining time shrinked must appear properly on the HUD of the affected player. Proper graphic effect is a plus.
- **Rewind time**: this feature is similar to the Tracer ability in Overwatch (example link: https://www.youtube.com/watch?v=LRXkkns6IRU). Player should be able to rewind his time without affecting others' gameplay. Information about the time rewinding should appear on the HUD.
- **Wall run**: this feature is similar to the Lucio ability in Overwatch (example link: https://www.youtube.com/watch?v=YH0ePK5ia50). Player is able to run and jump on walls for a specific amount of time.
- **Drop weapon**: a killed player should drop his weapon with the current amount of ammo remaining. This pickup must remain in-game for a certain amount of time and then disappear.

Notes for Multiplayer

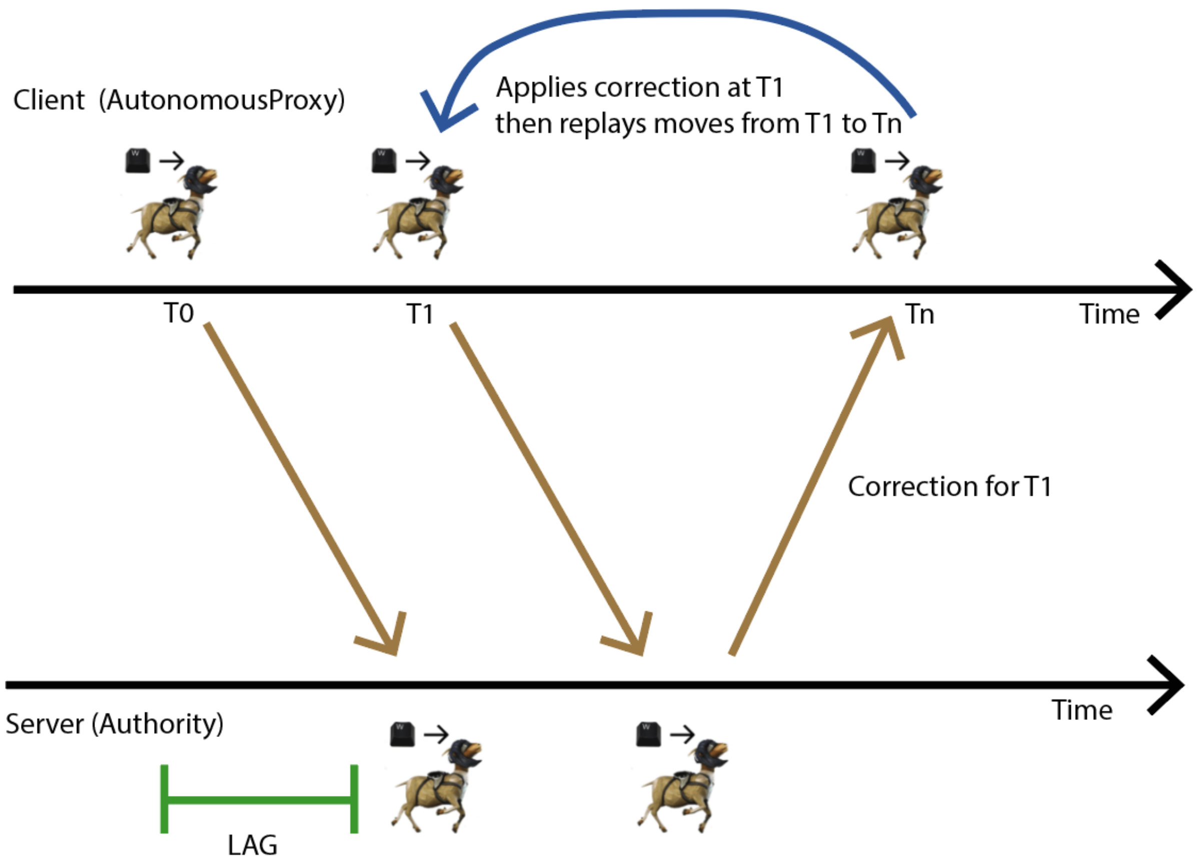
All of the above must properly work in multiplayer.

Prefer, wherever possible, a **responsive** implementation: when you press a button, you should see the result immediately; when the server disagrees it might correct you. To achieve this, you should implement a mechanism of **client prediction:** all clients immediately apply the inputs to the local physics simulation while simultaneously sending the same command to the server.

Clients periodically send to the server their local state marked with a timestamp. The Server may send a correction to the clients when their prediction error is too big. In this case the Server sends a correction request to the client marked with the timestamp of the wrong client state. Client, which maintains a queue of past moves, applies a correction backward in time, then reapplies all the moves up to current time (**server-reconciliation**).

When you implement client prediction, bear in mind that the CharacterMovementComponent is storing a list of past moves to be reapplied in case of a server error correction request (**server-reconciliation)**. If you don’t add your special moves to the CharacterMovementComponent the movement replay may not work properly (check it by simulating a 500ms packet lag).

The following diagram shows this mechanism.



References

Here you can find more info about this topic:
https://www.gabrielgambetta.com/client-side-prediction-server-reconciliation.html