Why should I know about QA

In 2022 Ubisoft cancelled a ton of games because no one liked them since July [Source]. Ubisoft as a company has a bad record for releasing bland, broken and buggy games that function so poorly on release that they leave players extremely dissapointed.

> *Whatever positive aspects existed in the game were tremendously overshadowed by how buggy, broken, and bland the experience was.*
>
> — *Peter Glagowski*
> *The Gamer*

So what can we do to make sure we don't end up like Ubisoft and make games that both work and are enjoyable?

Bland Games

This one is generally more up to your designers than to you guys as programmers but there are a few points I want to re-iterate.

From a business perspective

- As early as you can with your product test it with your target audience. You probably have some prototypes to see if something is fun.
  - Test if it's fun with the people you think are going to play your game
- What might be cool for your vision to your team might not interest people outside your team.

Bland Games

Some recommendations from Get some taste! Let's stop making bland games!

1. Figure out what you like about the pieces of art that inspire you
   - Practice by listing out what your inspirations made you feel is your current prototype emulating that
2.

Deliberately debate someone on something that you like and you know the other person doesn't. Figure out what might be a barrier to one person but is inconsequential to yourself. What is exciting to you but not all that cool to someone else.

- See how common those views are with your target audience.

3. Go into a thrift store (boot sale) and find three things you like. Why are those things immediately appealing to you. Who do you think owned these things why did they buy them, why didn't they keep them.
    - Bit of an odd one, but I think the exercise is to help you think critically about what makes something valuable to someone.

Buggy Games

So a lot of what you guys are doing now when it comes to making games and reducing the number of bugs comes down to how well you guys are architecting your code an how good you are at fudging solutions when things go wrong.

If you aren't already I assume you guys aren't against just dedicating sprints to solving the biggest problems in your games and having that be your solution.

How to use agile to improve feature quality.

1. One thing that I've seen in software outside of games is way more in depth **user stories** and requirements of each task.
    - Hopefully this is obvious: after you've implemented a feature make sure it still does what it's meant to do.
2. Does your feature break under circumstances that you can see it being used for?
    - If your feature isn't going to work everywhere make sure it warns the user of that. Stop it being used where it won't work. A feature shouldn't be considered complete if it breaks this rule.
3. Make detailed notes of any bug that happens on the spot.
    - If you have a bug tracking system put it in there. I think github can have issue tracking enabled on any repo.
        i. *Specific* — Note down what you specifically think caused the error.
        ii. *Detailed* — Include enough information to reproduce the glitch if possible.

Some notes on sprint reviews and planning.

The Backlog is your friend

If a feature does not meet the criteria of done at the end of a sprint, **It goes back in the backlog**.

Technical debt is horrifying

Avoid technical debt like the plague on any longer software project. It's just not worth the headache of major rewrites. And if you get to the point of not being able to work on your project without them in an indie environment that can be when interest dies in your project.

Bad Agile is worse than No Agile

No agile is better than bad agile. Everyone has to be using agile properly for you guys to reap the benefits from it. I'm sure some of you have trauma towards agile because of this already, you will have issues if you can't use it properly in the industry where it's in use. Especially with dedicated QA teams.

Break and Task

1. Look over your completed tasks for your projects right now,
2. is there anything in there that should be moved back to the backlog?

Test Driven Development

> *By thinking about the outward interface of the functionality first, it helped me get in a mindset of creating exactly what I wanted.*
>
> — *Alistair Doulin*
> *GameDeveloper.com*

You may have seen most software written outside the games industry is written with unit tests that are used to make sure that every feature works. Just because you're in games doesn't mean you can't do this if you're creating complicated systems that need to interoperate with each other.

Test Driven Development (TDD)

Is a development style that has the developer create automated unit tests for the component before writing the component. This is often done to ensure the quality of the functionality of the written components.

Why Test Driven Development?

> *Unit tests can be written to test all areas of a game from gameplay to rendering engines and anything in between.*
>
> — *Alistair Doulin*
> *GameDeveloper.com*

1. **Find out when code breaks.** — If you make a code change that breaks something that was previously working (and is tested) you will know about it immediately.
2. **Forces modular code.** — For code to be "unit testable" it must be modular with clear boundaries and "separation of concerns" between various systems.
3. **Allows you to “find the fun”.** — Once you’re guaranteed that certain requirements are met, you’re free to experiment with gameplay to make it more enjoyable without breaking the game. This also extends to giving designers more freedom when scripting. Encouraging experimentation by designers without programmer intervention is invaluable.

How to use unit tests for games

1. Design the unit test for the new feature, testing the outward interface seen by the rest of the system
2. Implement the feature as quickly as possible, cutting corners as needed so the feature is playable in the quickest time possible
3. The team tests the feature, makes sure it’s working and fun and we make iterative improvements to it
4. Once happy with the feature you can then optionally refactor the code to clean up any shortcuts and make it the optimal solution

As you can see this style of development could quite easily slide into an agile style timeline. Though it’s not the quickest way to develop. If you’re going to use this tool use it after the prototyping stage.

Prototyping however can inform you what will likely be the most powerful pain points for systems you need to create and can inform you what components might most need to be created with unit tests.

Seamless testing tools

If you do use unit testing in your projects you will likely want to make sure that these tests are run automatically at some point so you don’t have to think about the extra steps of running the tests yourself.

There are two ways to do this

1. CI/CD — Run the tests automatically when you push a commit have github tell you if the tests passed
    - Github's CI/CD
    - Jenkins — Good for keeping your CI/CD separate from version control
2. IDE/Engine plugins — Run the tests in your client as you develop your project.
    - NCrunch — Good for C# tests that run as you code in VS
    - Unreal's testing suite

Suggested reading/watching

- Test Driven Game Development Experiences
- Quality Assurance: A Methodology for Wide-Spectrum Game Testing
- Quality Assured: What It's Really Like To Test Games For A Living

Suggested activities

Practising Q&A

- Choose a game (one of your own, a friend's/fellow student's, or any "less polished" released title) and play it as if you were performing QA, making notes of anything you notice that seems odd.
- Pass your notes to your neighbor to see if they can reproduce the same effects!

Learning About Unit Tests

- Choose a code project that has some features that have broken after some changes and create a unit test for this feature.

For those of you with dissertations you will need to test your artefact later on down the line to meet the criteria. So spending the time to learn how it works now might take some pressure off down the line.