Software Engineering Conference Russia 2018

October 12-13
Moscow

A Comprehensive Approach to Quality Assurance in a Mobile Game Project

Maxim Mozgovoy

The University of Aizu
Helium9 Games

mozgovoy@u-aizu.ac.jp

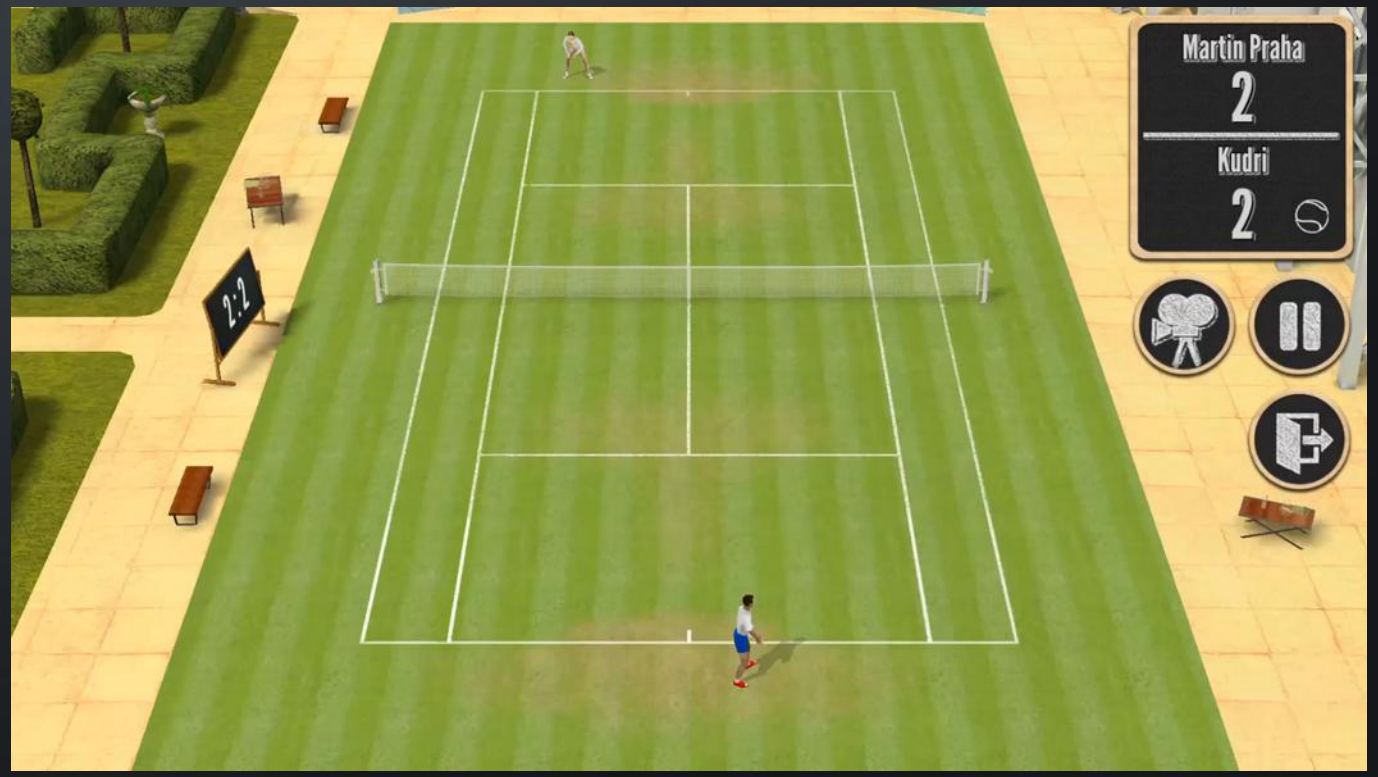
Evgeny Pyshkin

The University of Aizu

pyshe@u-aizu.ac.jp

World of Tennis: Roaring ’20s

- Developed by Helium9 Games (www.worldoftennis.com)
- Released in late May 2018
- 100 000+ installs on Android
- Over 1800 user reviews (avg score is 4.5)
- Similar figures for iOS and Windows Store



What's Special about Gamedev?

- Heavy reliance on unstable 3rd-party libraries and tools.
- Diversity of hardware and software platforms.
- Abundance of visual and sound issues.
- Large proportion of high-cost unit testing code.
- Deep integration of GUI and animation.

Tool #1: Crashlytics

Embed Crashlytics crash reporting service into the app.

(Fortunately, the game requires internet connection, so the crashes can be reported automatically during the next session)

Ex. outcome:

Identified devices with insufficient RAM to run the game.

Tool #2: Autobugs and Manual bugs

Use reporting assertions and manual reporting tools

Every time there is an exception or assertion failed cause, the game reports it to us. Also, the testers and the users can report us a bug any time via the game menu.

Ex. outcome:

A massive amount of bugs were reported this way.

Tool #3: Manual Testing

We (obviously) test manually a lot

Our testers follow a certain procedure (what they test).

They also record their actions on video, so we can later examine their reports.

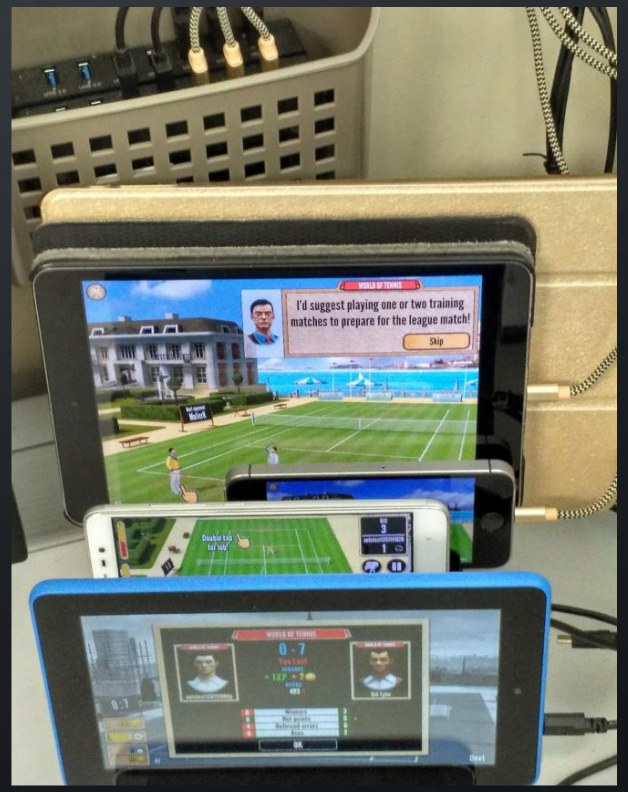
Ex. outcome:

These tests were mostly useful for finding visual glitches and subtle bugs (such as wrong prices or incorrect calculation of player rankings).

Tool #4: Automated Smoke Testing (1)

We set up an auto-testing device farm

Test Servers
(Appium + our software)



Test Devices
(Android, iOS, Windows)

Tool #4: Automated Smoke Testing (2)

Every time there is a new build in the system, the farm perform a set of tests (taking around 2 hours) that ensures the basic functionality of all subsystems.

In particular, it can:

1) Pass game tutorial.
2) Customize the player and upgrade his/her skills.
3) Play a training game.
4) Play a set of league matches.
5) Link a Facebook account to the game.

Tool #4: Automated Smoke Testing (3)

This is the most advanced system we use.

Ex. outcome:

- Autotests generate crashes and autobugs!
  So we get a cumulative effect of using the systems together.
- Autotests free our testers from checking basic game scenarios.
- They also report indicators such as FPS, free memory, etc.
- They can work as stress tests, checking the ability of the application to survive long (3-4 hours) game sessions.

Conclusions

- Game development process is hard on QA, as many established practices such as refactoring, unit tests and automated functional tests are not always easy to implement.
- QA should be taken pragmatically: some bugs are beyond our control, so we've to live with them and alleviate their impact.
- Automated bug reporting facilities is a must for a game.
- Automated smoke / GUI tests are extremely helpful.
- The tools we use exhibit a clear cumulative effect, so they provide a reliable safety net for the game when used together.