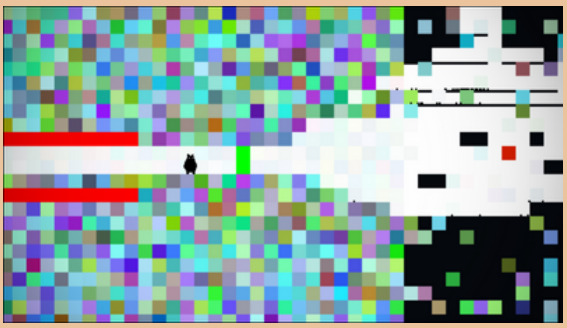
Programming for Humanities Students

Brianna Blackwell

When you ask a student in the humanities how they feel about programming, they will often say they find it daunting. In the digital humanities, however, being able to write or understand code is important in a significant amount of the work in the field. My question, then, is: How can we introduce programming to humanities students in a way that is both unintimidating and relevant? This question is important to me because I struggled (and often still struggle) while learning to program. The bar of entry to a lot of digital humanities work is far too high, and this turns a lot of otherwise interested people away from the field. I was lucky enough to have some background in programming before I started, but for those who don't, many existing resources can be alienating. They are often geared towards math and the sciences, are only available in courses or workshops, or are on sites like github, which are less likely to be frequented by humanities students. My approach to this problem is based off of what I would have found most helpful when I first started to learn programming: a game that teaches programming while embracing exploration and failure.

Inspiration and Insights

When thinking about how to approach a programming game for humanists, I've drawn inspiration from a few places. Most important have been the games *Loved* by Alexander Ocias and *Hades* by Supergiant Games. *Loved* is a browser-based platformer that explores (among many other things) the relationship between a game's instructions and the player's free will. The more the player disobeys the narrator's instructions, the more colorful the game's world becomes. Using this game, I have been thinking about how to push the player to explore the bounds of a programming language in a way similar to the way they would explore a game world. For me, the colors in *Loved* have been a really helpful metaphor for responding to this sort of experimentation.



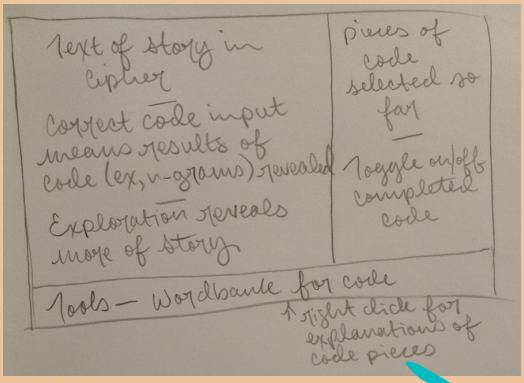
Loved by Alexander Ocias

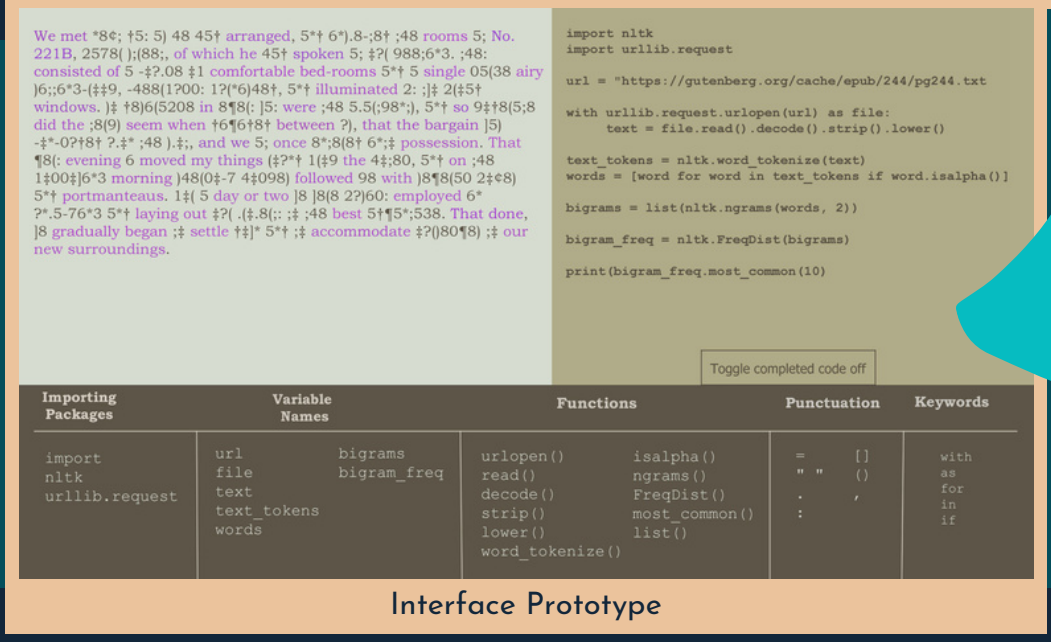
Hades by Supergiant Games

Rewarding the player's experimentation when they succeed isn't enough, though. Programming can be really frustrating, especially for beginners. The error messages that result from trying to run code that has logical or syntactical problems are both frequent and cryptic. Enter *Hades*, a rogue-like game that thrives in a contentious genre because the creators figured out how to make failure as compelling as winning. When the player fails a run, they get to experience more of the game's story and characters. From *Loved* and *Hades*, I've gained a lot of insight into how to encourage the player to explore and learn, and how to encourage them in both the success and failure that experimentation requires.

Design and Interface

The design of the game has evolved quite a bit. Originally, I had planned to make a narrative platformer. However, I found it difficult to integrate both programming information and rewards into the game while keeping the focus on a story. Instead, I've decided on a codebreaking game that rewards experimentation in code by revealing pieces of a story. Alternatively, players who already know how the code works can run the completed code and reveal only the parts of the story that would be produced by the code.





Interface Prototype

In this interface prototype, the story in the upper left begins as encoded text. The player can interact with the wordbank on the bottom, left-clicking to input code into the workspace on the top right or right-clicking to learn more about a piece of code and how it works in the completed program. Interacting with the wordbank also decodes words in the story. If the player has put code into the empty workspace, they can run it at any time, and an explanation of what that code does or the error messages it produces will appear. A completed version of the code can also be toggled on or off in the workspace. When on, the player can run the code to highlight the words in the story that would result from that code. For example, the code in the prototype above finds the most common word pairs in the given text. If the player runs the completed code, the word pairs would decode or change color, depending on whether the player had already revealed the words by interacting with the wordbank.

Lightning

The very kind and thoughtful advice I've received while sharing this project during Lightning at Building 21 has been incredibly helpful. A lot of people have had a significant impact on my thinking about this project and the direction it has taken so far. Despite the fact that I haven't been able to make it as often as I'd like to, Lightning sessions have always been welcoming and encouraging spaces to think through these questions and learn from a community.