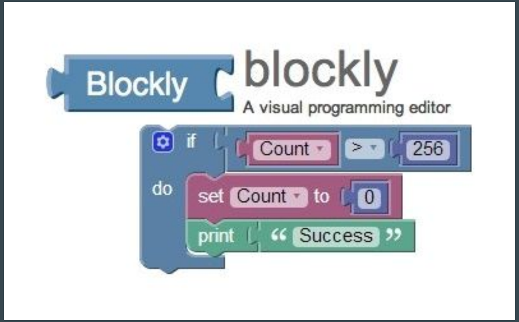
THE BIT BANGERS

Banging on bits since 2017

Project Description

Create an HTML5 game utilising the Blockly API to aid in teaching children and young adults the basics of programming.

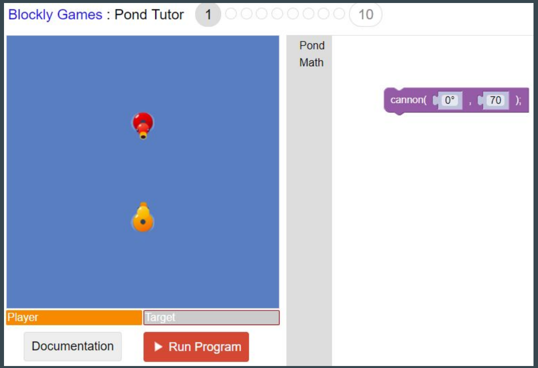
Target Audience: Kids in junior high or high school

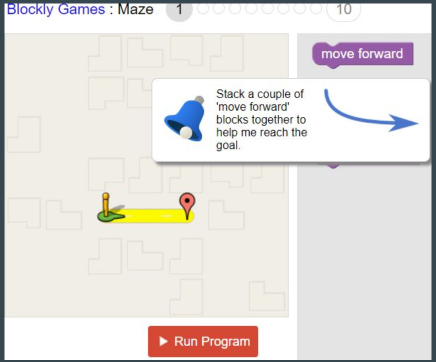


Project Requirements

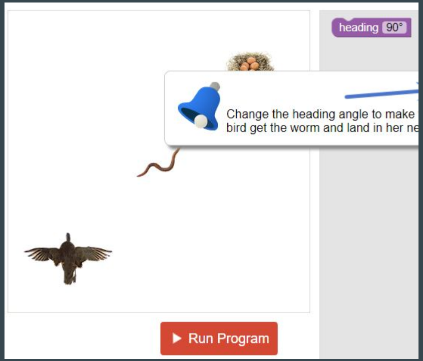
- Multiple levels -Complete (Game has 5 levels)
- Levels makes use of programming concepts to teach programming -Complete (Use Blockly Code Blocks to teach code)
- Single Player -Complete (Every level is single player)
- Graphics/Sprites?Animations -Complete (5 Maps Completed, Engineer sprite, platform sprite, gate sprite, star sprite, Aleksey sprite completed.)
- Title Screen -Complete (Complete and looking rad)
- Draggable Program Blocks -Complete (Blockly code implemented in blocks)
- Instructions -Complete (Basic instructions given in early levels)
- Level Transition -Complete(Levels transition to next level in sequence)

Differences between our game and others in domain:





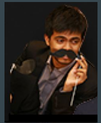
Stack a couple of 'move forward' blocks together to help me reach the goal.



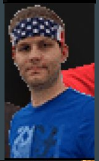
Change the heading angle to make bird get the worm and land in her ne

- More visually appealing
- Works like many 2d games kids have already played (Mario, Sonic, etc...)
- Has instructions to teach kids programming concepts such as If statements and For loops.
- Has a rad title screen
- Has an awesome robot

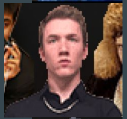
Bit Banger Responsibilities:



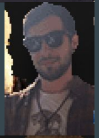
Rahsin Mulk - Sprite Design, Map Design, Gate and Platform controls, Blockly Code



Jonathan White - Map Design, Document Deliverables



Harrison White - Backend, Title Screen, Cool Music



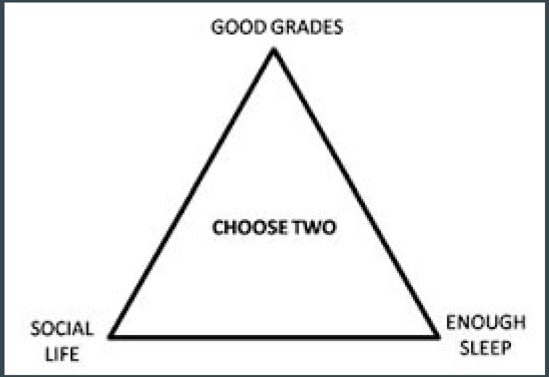
Kyle Christiansen - Blockly Code, Game Interface, Robot movement controls, block code chaining design

Bailey Miller - Level Instruction, Camera Movement, Scribe

Bit Banger Performance Evaluation

Our team at first struggled to maintain balance between working on this project and making time for other classes. We also had many of our team members either working, or living in Cedar Rapids, which caused difficulties in meeting up. Despite these initial setbacks our team pulled together to make a fun, functional, and educational game with a bit of flair.

Performance Evaluation Ranking: Bit Banging Bosses



Development Methodology

Last Minute Panic/Agile

Rough version of agile development

- Utilized Facebook messenger to keep each other up to date and schedule meetings
- Listed “tickets” and assigned roles and due dates for the week
- Considered “rough” because unlike a real agile team, we had to coordinate around busy school schedules and full time jobs.
- Ended with a mighty last minute push.

Best Practices

- Source Control Github
- Continuous Integration(Cultural Component) We would routinely commit code changes into our “blocka” repository after every change we made. We would then manually test them using the maps we created.
- Issue Tracking We used Facebook Messenger to track all of our issues. Whenever people were struggling with something that would take more than 15 minutes to fix, we would add it to Facebook Messenger for the team to follow.
- Test Driven Development We determined that developing automated tests for our game did not have much of a practical use. Almost everything that we did could be more thoroughly tested manually.
- Refactoring Condensed some areas of code that were repeated into functions. This still needs some work, but we did make a start.
- Code Reviews We held code reviews not as a whole team, but generally in groups of 2’s. Ex: Jon created maps and then held a review with Rahsin to make sure they were implemented in the same manner.
- Postmortem After our first team review we discussed how things had gone so far and what we could do to make it better. We also collaborated on a post mortem before starting this demo.
- Continuous Development We did not have enough iterations of this product to truly make use of this best practice.

Postmortem

What went right:

- Learned quite a bit about javascript in general.
- We made our general idea become a reality.

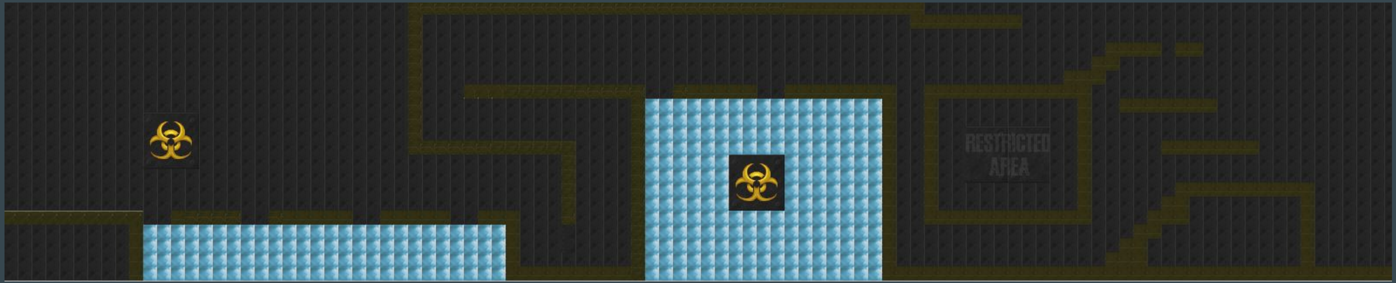
What went wrong:

- Didn’t get everything we wanted implemented.
- Code cleanliness

What we could have done differently:

- Think about how different Apis will interact early on rather than wait for something to go wrong.

Highlights



Interesting Levels

Engineer Sprite

Team Morale

Robot Moonwalks when walking backwards

Successfully learned the Fundamentals of Software Engineering