Escaping the Turing Tar-Pit with AI Programming Blocks

Alexander Repenning
School of Education
PH FHNW
Windisch Switzerland
alexander.repenning@fhnw.ch

ABSTRACT

Game design is often considered a motivational approach to get young children interested in programming and computational thinking. However, while the idea of game programming may be compelling from an educational point of view, creating games with interesting interactions that are actually fun to play remains challenging. Modern tools aimed at novice programmers should empower their users to create games, such as Pac-Man, that approach or even exceed the gameplay of 1980's arcade games. By adding a high-level AI pathfinding block to the RULER.game tool, 13 students in grades 1-4 attempted to build Pac-Man-like games. The findings suggest that all students were able to create Pac-Man-like games with compelling gameplay interactions, including ghosts finding the shortest path through complex mazes to Pac-Man, multiple ghosts collaborating with each other, and sophisticated game world topologies featuring toroidal portals.

CCS CONCEPTS

• Applied computing~Education~Interactive learning environments

KEYWORDS

Computer science education, artificial intelligence, computational thinking, mobile programming tools, programming for children

ACM Reference format:

Alexander Repenning, 2024, Escaping the Turing Tar-Pit with AI Programming Blocks, in *Proceedings of the 19th WiPSCE Conference on Primary and Secondary Computing Education Research (WiPSCE '24), 2 pages*. https://doi.org/10.1145/3677619.3677646

1 The Turing Tar-Pit

Especially in early K-12 grades computer science education often prioritizes affective goals such as getting students excited about programming, over cognitive goals such as learning important principles of programming. In these contexts the creation of personally interesting artifacts can play an important role. Provided childrens' nearly universal interest in video games, game programming has been considered particularly low-hanging fruit. But what if programming a game turns into a tedious coding exercise producing a non-compelling game? There is the potential danger that students' initial negative perception of programming, e.g. "programming is hard and boring" actually gets reaffirmed.

ACM ISBN 979-8-4007-1005-6/24/09. https://doi.org/10.1145/3677619.3677646

Alan Perlis introduced the Turing tar-pit [2]: *"Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy."* The Turing tar-pit describes potential tension between cognitive and affective goals of game programming in schools. The cognitive goal may be to learn computational thinking (CT) through the process of designing and programming a game. The affective goal may be to create an exciting game that game creators and their friends may actually enjoy playing. Ideally, these goals are aligned.

Even simple 1980 arcade style games such as Pac-Man or Frogger have some compelling gameplay characteristics that may be hard or, practically speaking, impossible to program for programming novices using tools such as Scratch. For instance, in a Pac-Man game even young students expect ghosts to exhibit intelligent pathfinding behavior and not just random movement. Using tools such as Scratch students are likely to fall into the Turing tar-pit producing complex code without actually creating a compelling game.

This research employs a Pac-Man-like game as a benchmark to explore cognitive and affective challenges. First the article illustrates how a group of 13 students, grades 1 - 4, are programming a "compelling" Pac-Man game using AI programming blocks added to the RULER.game computational thinking tool. Then, the article tries to deconstruct the necessary combination of abstractions and affordances to escape the Turing tar-pit.

2 Falling into the Turing Tar-Pit

13 students from 1st to 4th grade used RULER.game [3], a highly accessible computational thinking tool aimed at young children with limited reading skills, to design and program a Pac-Man game in 90 minutes. RULER.game works on mobile devices (Figure 1).

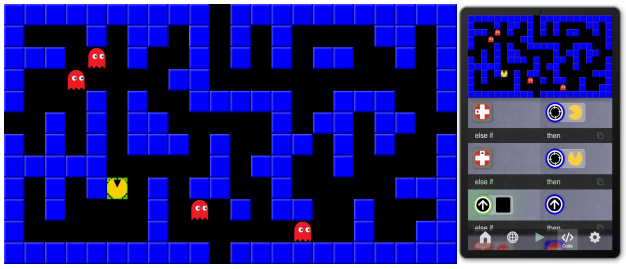

Figure 1: Left: World designed. Right: iPad showing code

Game design started with a class discussion of the Pac-Man game breaking down the Pac-Man world into five basic shapes: Pac-Man, ghost, wall, floor and point. Based on these shapes students designed their own levels (e.g., Figure 1). Designs varied widely. The level of complexity of these levels, that is the number of rows and columns, or the inclusion of portals (vertical or horizontal world wrap around points) was somewhat correlated to the age of the students. Some of the 1st graders designed simple 4 x 7 worlds.

Programming started with Pac-Man. RULER.game has a up/down/left/right directional pad (D-pad). One approach to control Pac-Man with the D-pad is to write four rules of this flavor: IF key is pressed THEN Pac-Man rotates according to the direction and moves one step in that direction. Students pointed out that it would make more sense if Pac-Man would keep moving in the direction that it is heading. Incidentally, this is also how the original game works. This approach requires four rules rotating Pac-Man and a fifth rule (green highlight in Figure. 1, right): IF Pac-Man sees the black floor ahead of itself THEN Pac-Man advances forward. RULER.game offers absolute grid references (called Bird View) as well as relative grid references (called Turtle View).

Now the Turing tar-pit was entered. Random movement of ghosts was not considered compelling game play. A first version of the ghosts was programmed adding a single rule moving randomly on top of the floor. It became clear that the ghosts have no real understanding of how to move towards Pac-Man in a complex maze. As anticipated, students quickly complained "but my ghosts are not smart at all, they should be trying to catch Pac-Man." The initial interest built up by the exciting prospect of creating one's own game was suddenly replaced by frustration.

3 Escaping the Turing Tar-Pit

A discussion exploring more compelling options revealed how a mouse would be able to find cheese inside a maze without being able to see the cheese. The formal notion of hill climbing [4] was approximated by the students suggesting that the mouse would head in the direction where the smell of cheese is most intense.

In order to escape the Turing tar-pit in RULER.game it was necessary to introduce new abstractions based on important affordances. The *moveRandom* action was replaced by the *advanceTowardsTarget* artificial intelligence (AI) block implementing Collaborative Diffusion [4]. For single agent search, Collaborative Diffusion works similarly to an A* algorithm to find the shortest path in a graph representing the maze. However, unlike A* Collaborative Diffusion also works for the much more complex multi-agent pathfinding. For instance, if there are n ghosts having n paths to Pac-Man they would split up optimally. That is, they would actually collaborate, leaving the Pac-Man no exit route.

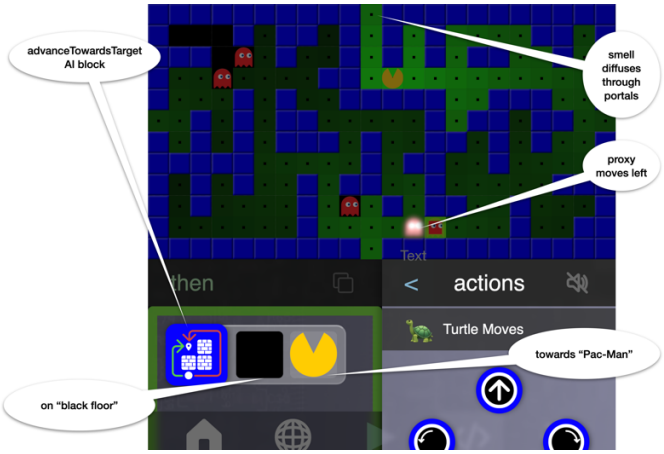

Figure 2: AI block (left, bottom) and visualization of smell

Figure 2 illustrates *Proxy-Based Programming* [3] in RULER.game. The user selects a ghost to enter a programming sandbox. A proxy object is created. After picking the advanceTowardsTarget action (parameter 1: on, parameter2: towards), the ghost proxy is animated to move to the left indicating what that action would do. In addition, the smell of the Pac-Man, i.e., Collaborative Diffusion value is visualized making it clear this ghost would move left. This may be somewhat surprising as the shortest path is through the South/North portal and not to the right to later head up.

The students quickly realized that the AI is "too" smart, making the game frustratingly challenging. Ghosts would find a way, sneaking sometimes surprisingly through the portals, to quickly circle Pac-Man. Fortunately, it is much simpler to dumb down a great AI than to smarten up a bad one. The students had to be shown the *setPlaybackSpeed* action making it possible to slow down the ghosts and to speed up the Pac-Man to the point where the game became playable and indeed highly compelling.

4 Affordances enable Abstractions

Analysis of numerous Pac-Man projects in Scratch/Scratch Jr. revealed a lack of compelling pathfinding AI. Most used predictable scripted or random movement. Attempts at pathfinding failed for all but the most trivial mazes. These projects typically had large code bases without working single or multi-agent pathfinding, and often exhibited common Scratch code smells [1] such as redundancies.

If a high-level of abstraction AI pathfinding block can be easily added to RULER.game why would adding a comparable block to Scratch be much harder or perhaps even impossible? A short answer is difficult to formulate. However, the key is that high-level abstractions necessary to enable novice programmers to exit the Turing tar-pit depend on specific affordances. In Scratch the addition of these affordances would probably require fundamental architectural changes:

- ***Grid:*** Enables topological parsing and automatic pathfinding working even in toroidal worlds including portals. Grid-aware blocks enable expressing complex behaviors.
- ***Class-Instance Object Models:*** Reduce redundancy and simplify code management for frequently used objects like floors, walls, and points.

5 Conclusions

Students as young as 1st grade were able to use a high-level of abstraction AI pathfinding block to create a compelling Pac-Man-like game with the RULER.game computational thinking tool.

REFERENCES

[1] Hermans, F. and E. Aivaloglou, "Do code smells hamper novice programming? A controlled experiment on Scratch programs," in 2016 IEEE 24th International Conference on Program Comprehension (ICPC), 2016.

[2] Perlis, A. J., "Special feature: Epigrams on programming," ACM Sigplan Notices, vol. 17, pp. 7-13, 1982.

[3] Repenning, A. and A. Basawapatna, "RULER: Prebugging with Proxy-Based Programming," in Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, Liverpool, UK, 2024

[4] Repenning, A., "Excuse me, I need better AI!: employing collaborative diffusion to make game AI child's play," in Proceedings of the ACM SIGGRAPH symposium on Videogames, Boston, Massachusetts, 2006, 169-178.