Creative Coding - Week 7

Game Design with PICO-8

Lecturer: Dr Michael Cook (mike.cook@kcl.ac.uk)

Figure 1: A screenshot of *Snek*, a thrilling videogame experience for all ages, made in PICO-8.

This week is the second of three weeks looking at game design. We're looking at PICO-8, which is a type of tool known as a *fantasy console*. Fantasy consoles are full game development environments, but they normally have self-imposed limitations that are similar to those experienced by game developers in the 1980s and 1990s. For example, the BBC Micro had as little as 16KiB of memory, a palette of between 2 and 8 colours, and only 100KiB of hard drive space. Every new sprite, sound effect, bit of text or line of code ate into these restrictions.

Last week we used PuzzleScript, which has some restrictions but they are mainly to achieve its design goals (for example, its rule system is restrictive but it is designed to be quick and simple to use). The one exception is its sprites, which are kept to 5x5 arbitrarily, but this makes PuzzleScript games look consistent and keeps the focus on prototyping. By contrast, PICO-8 chooses restrictions to constrain and focus you, the designer. It has a palette of 16 colours, a screen resolution of 128x128 pixels, 4 channels for sound, and the code for a game must be less than 65,535 characters. Many developers use it as their main development environment, for prototyping, education, game jams or sketching.

It might seem strange to voluntarily take on these restrictions, when you could just download a tool like Unity or Unreal and make 3D games with no limits. I think there are a few reasons:

Creativity The phrase 'constraints breed creativity' is something people mention a lot in relation to making games. By restricting what you can do, you're focused on the remaining opportunities. It can help people come up with creative workarounds to problems, or explore problems they wouldn't normally.

Simplicity A major problem for new game developers is choosing a game idea that is too big. PICO-8 games can get pretty complex, but the tool's simplicity encourages us to make smaller, simple games, and actually finish making them. This is really valuable.

Challenge For a lot of us, these days there isn't much need to be efficient when making software. Software, apps and websites chew up memory and resources and don't really think twice about it. PICO-8 forces us to think about how to be efficient, with our code, with our art, with our music. Can we reuse this sprite somewhere else? Can we write this bit of code with fewer lines? It's great practice for being a more efficient and effective developer (if this doesn't appeal to you, don't worry - most PICO-8 developers don't hit these limits!)

Portability PICO-8 games are very portable. In fact, an entire game is stored inside a .png image file showing the game's front cover. PICO-8 itself runs on anything, including a lot of small handheld devices, and every game runs in the browser, too. Every PICO-8 game is open source by default (you can see how great designers make games with it!) The PICO-8 app contains a code editor, music editor, SFX editor, sprite editor, map editor and more in one package, plus a browser for finding new games. It's a very light, fun and open ecosystem to be a part of.

With that sales pitch out of the way, let's crack on with today's lesson!

1 Getting Started with PICO-8

PICO-8 is paid software. Right now it costs $15, however there's also a free version that runs in the browser. **If you're in the session with me for our code class, I have some codes for free copies of PICO-8, please ask me for one!**

Click here to open PICO-8 Education Edition

A major difference between this and the normal app is that you can only save your work by downloading it. **DO NOT CLOSE THIS TAB WITHOUT SAVING YOUR WORK.** If PICO-8 doesn't launch automatically, click the triangle in the middle of the screen. You should see a brief intro sequence and then be presented with a terminal.

1.1 Basic Commands

PICO-8 starts you off at a terminal. This works mostly like a Linux terminal, you can type various things here:

- `help` – prints a list of commands.
- `ls` – lists the contents of the directory you're in.
- `cd <dirname>` – can be used to change directory (you generally won't need to do this).
- `save <filename>` – saves the current game to a file called `<filename>`. In the Education edition, this immediately downloads the file as a .p8 text file that contains the whole game (including art).

Figure 2: PICO-8's two modes: system (left) and editor (right). Press ESC to toggle between them.

- `load <filename>` – loads the named game. In the Education edition this instead opens up a file browser where you can select the file you want to load. We'll use this in a bit.
- `run` – runs the currently loaded game.

1.2 Changing Screens

PICO-8 has two different modes. You start in the console mode, which is where you load, run and save games. The other mode is the editor mode that contains the code, sprite, music editors and other tools. Press `Escape` to switch between the two modes. Figure 2 shows screenshots of the two different modes.

In the editor mode, you start in the code view. This is where you write all the logic for your game. For the educational version, you have to use this editor, but on desktop you can use your own editing tools which many people find more comfortable. The editor has several other tabs:

From left-to-right, these icons are: Code Editor; Sprite Editor; Map Editor; Sound Effect Editor; Music Editor. We're going to mainly use the Code, Sprite and Sound Effect editors today. Clicking on the icons switches between them. **Click the Sprite Editor (the second icon).**

1.3 Sprite Editing

Figure 3 shows a view of the PICO-8 Sprite Editor with different parts annotated. You can click in the bottom area to select which part of the sprite map to draw into, and then draw a sprite in the window in the top-left. In the exercise for this week we'll give you some sprites to start off with.

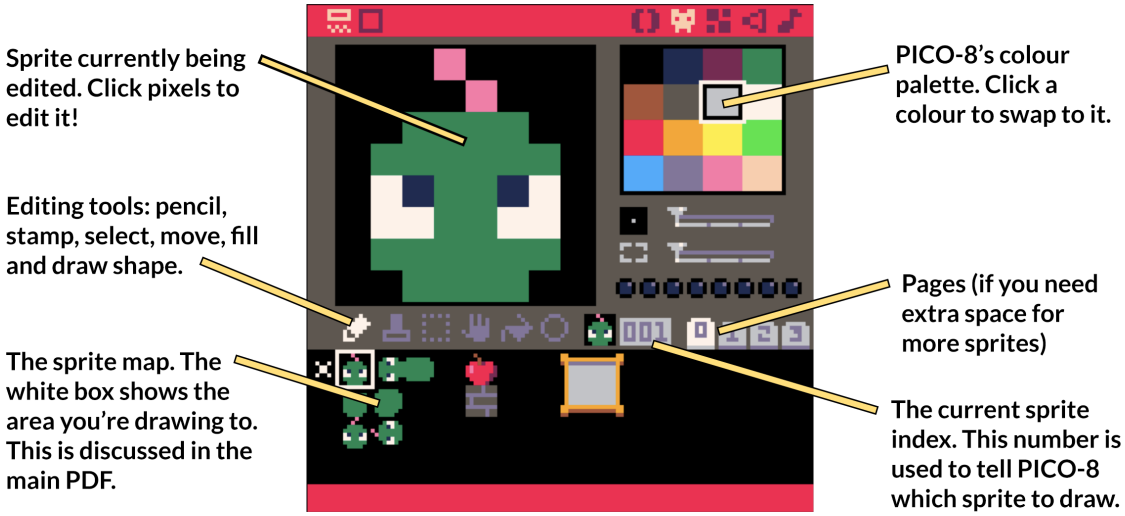

Figure 3: An annotated view of the PICO-8 Sprite Editor.

The most important thing on this screen is the **sprite index**, which is marked on the right-hand side of the editor. This number is unique for each sprite. When you want to draw a sprite on the screen, for example, you'll tell PICO-8:

```
spr(1, 32, 32)
```

This tells PICO-8 to draw the sprite with index 1, at the co-ordinates (32, 32) on the screen. We use the index in all sorts of places.

There are lots of other cool features here, like setting flags for sprites, or drawing big sprites next to each other that we can chop up and tile, but they're beyond the scope of today's lesson. **Click on the fourth icon to go to the sound effect editor.**

1.4 Sound Effects

Figure 4 shows an annotated screenshot of the sound effect editor. This works, in some ways, just like drawing a sprite. You can pick a waveform to paint with on the top row (each waveform has a different texture/sound, a bit like picking an instrument), then click and drag to paint notes, or click one by one to adjust. Higher lines equate to higher pitches. To preview a sound effect, press spacebar and it will play.

Just like sprites, we need the sound effect index (the number in the top-left) to play a sound effect. It works like so:

```
sfx( 1, 2 )
```

This tells PICO-8 to play sound effect 1 on channel 2 (PICO-8 has four sound channels, meaning you can't have more than four sounds, including music, playing at once).

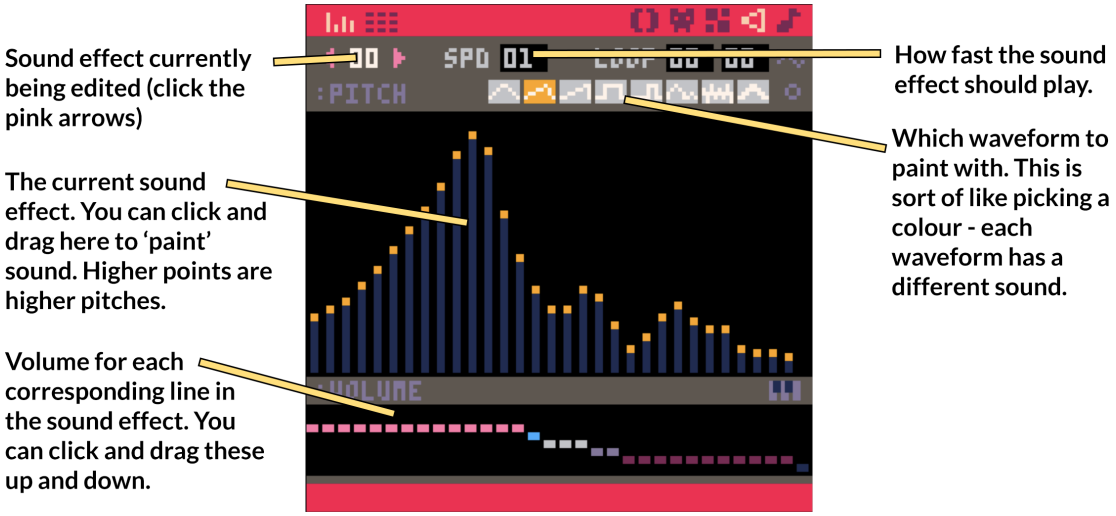

Figure 4: An annotated view of the PICO-8 Sound Effect Editor.

1.5 Other Screens

PICO-8 has two other screens: one is the map editor. The map is a special data structure that can store sprite layouts, and it lets you do things like design levels and maps with the mouse rather than writing code. It also has some special game logic attached to it that makes it easier to check if objects are present. The example game for today uses it a little bit. You don't have to use it, but it can make some things easier.

The final screen is the music editor. The music editor has some overlap with the sound effect editor, but is more focused on making longer, looping performances. There are some great tutorials for this, I've linked some at the end of this PDF.

One really useful resource is **The PICO-8 Cheat Sheet**.

2 This Week: Snek

Today we're going to practice designing a new game feature, and thinking about different parts of that process. We're going to start by playing a game, thinking about what we'd like to change about it, and then designing a new feature. We'll make art, a sound effect for it, and then code it into the game. It's going to be a real crash course! PICO-8 is a big tool, and to learn it properly you need to take some time on your own to work through a guide. But today will give you a taste of it, and get you thinking about the design process.

The game we'll be using is Snek, a clone I made of the famous arcade/mobile game Snake. You can download the files for the game from the same page you got this PDF from. The file will download as a .png, because PICO-8 exports all of its carts in a single image file. Save the file somewhere on your computer, then open up PICO-8 in your browser.

Figure 5: Screenshots of Snek, Game of the Year 2026 contender.

2.1 Loading and Running Snek

If you are using the browser version of PICO-8, type `LOAD` into the console and press enter. The browser will pop up a file selection window. Browse to the Snek file you downloaded, and select it. PICO-8 should tell you the file has loaded successfully. To test it, type `RUN` and press enter.

You should now see the main menu. Press `Z` to start playing, and use the arrow keys to move. Avoid hitting the walls or your own tail, and eat as many apples as you can. When you're done, hit `ESC` to bring up the console, and then hit `ESC` again to open the code editor.

2.2 Task 1: Familiarising Yourself With PICO-8

Your first task is to make a few small changes to Snek to get used to working with PICO-8:

- **Change the speed of the game.** Snek works by counting each frame, and then calling `Tick()` when the frame count hits a target number. `Tick()` moves the player forwards and checks for things like eating apples and so on. Try to find a variable you can change to make the game run faster.
- **Redraw one of the sprites.** Go to the Sprite Editor tab and pick one of the sprites to redraw. I recommend redrawing either the wall sprite, or the apple. The Snek is extra work (it's four sprites in total).
- **Add a sound for the apple being munched.** Go to the Sound Effect Editor tab and make a sound. Press space to test it. A simple 'boop' should be okay. Then, find where the apples are eaten in the source code, and call `sfx(0)`. Play the game again to test.

Remember, **to test the game** press `ESC` to return to the console, type `run` and press enter.

2.3 Task 2: Adding A New Feature

I once heard someone say that a game design isn't about having a really big idea, it's about having ten thousand very small ideas, and figuring out how to sew them all together. A game design is a series of interesting decisions, and each choice you make affects all the others. Today you're going to add something new to Snek, and change the game design into something of your own.

Create and add a new object to the game. It should change the way the game plays in some way. You can make any change you want, but I recommend either:

- Add a **powerup** to the game - something the player can eat that makes the game easier.
- Add an **obstacle** to the game - something the player must avoid, or that makes the game harder somehow.

Remember what I said last week: keep your idea small. Small ideas can be finished, tested, improved and polished. To give you some guidance, here's a checklist to go through as you think about something to add:

- First, draw a new sprite to represent whatever object you're going to add. Draw it in a new part of the sprite map, and make a note of its index.
- Next, think about when you want it to be added to the game, and write some code to do this. For example, apples are added to the game on a timer. You could create a new timer for your object, or add an object every time something happens - like every time a fruit is eaten.
- Now think about what rules are associated with your object. Can it be eaten like an apple? What happens if the Snek hits it? Look through the code for examples of existing game logic and see if you can implement your own rules for your object.

This is the hardest task I've set in the class so far – please ask me for help or advice if you need it.

2.3.1 Ideas

If you're stuck thinking of what to add to the game, I've included some ideas here. Maybe you can mix some of them together or maybe they inspire an idea of your own!

- A **lemon** that acts like an apple, but when Snek eats it the game runs twice as fast for ten seconds.
- An **orange** that is scared of Snek and will run away (until it hits a wall, or the Snek's tail).
- A **poisoned apple** that turns the Snek's tail to stone (the Snek goes back to having no tail, but the current tail tiles turn into stone blocks).

If you're adding new fruit, like the ideas above, you might want separate timers, or to have a percentage chance that it is spawned instead of an apple (or at the same time). Indie studio Punkcake Delicieux have a great Snake-like game called Serpentes, which you can find here (paid - but you can also find videos of it being played online).

2.3.2 Alternative Tasks

Today's main objective is to get used to working with PICO-8, so if you don't feel like coding you could try experimenting with another part of the tool instead! Here are some alterate tasks:

- Create some music for the game - maybe a main menu music, some music for gameplay, and then a music sting for when you die. A YouTuber called Gruber has a great tutorial series about this.
- Create a new sprite set for the game, perhaps following a different style of art. Look online to find some good examples of different PICO-8 art styles!
- Animate the sprites. This requires a bit of extra code, but it's not too hard - the trickier part is drawing the sprites themselves.

2.4 Important Sections of Snek's Code

I've tried to comment the code extensively, but here are some important parts to check:

- `update()` is called thirty times per second. We check for player input here.
- I use timers to see when to add fruit or move the player. These times are also handled in `update()`.
- `draw()` is also called many times a second, whenever the screen needs updating. Most of this code doesn't need to be touched.
- `addfruit()` shows you how I add fruit to the level whenever the timer is hit. I do this using the map functions (`mset` and `mget`).
- `tick()` is where I move the player, and also where I check to see if the player has hit fruit (good) or the wall (bad) or their own tail (also bad).

If in doubt, just ask me.

3 What Next

3.1 Share your work with me!

I'd love to see what you make, and I'd also love to be able to share people's work at future department events, or show to students in future years for inspiration! I would love it if you shared what you make - anonymously if you prefer, or with your name for full credit.

REMEMBER: PICO-8 DOES NOT SAVE YOUR WORK IN THE BROWSER. To download your work, you MUST type 'save' into the console. If you close the tab, your work is lost forever.

Ways you can share your work:

- Email the source code or your exported PICO-8 card to me (mike.cook@kcl.ac.uk)

- Post it to GitHub and share with me (username: gamesbyangelina)
- Drop it anonymously in this form: https://forms.gle/qh8UWnqW1fNJyfxz8.

I recommend registering for an account on PICO-8's website as well, as it's useful for publishing and sharing your games.

3.2 Further Reading

PICO-8 is an extremely popular game development environment and there are a lot of dedicated resources teaching you how to make games with it.

- Nerdy Teachers have a highly-regarded tutorial series.
- This YouTube tutorial by Lazy Devs is also often recommended.
- I'm a big fan of the PICO-8 zines here.
- Gruber's PICO-8 Music tutorials are great and taught me a lot.

Also remember that **every PICO-8 game is open source**. This is a huge deal! You can download any game you like and see how it was made. Some games I recommend:

- Celeste – This game was developed into a full game and released on every platform. It's one of, if not the, best game of the 2010s. This is the version that inspired it!
- Solitomb – A dungeon battler, but it's Solitaire. Very cool, the last game this guy made in PICO-8 became a hit indie game.
- Make Ten – Incredibly simple, sharp and addictive puzzle game.
- Mystic Realm Dizzy – In the early 1990s there was a surreal game series about playing as an egg with arms and legs and a face. This indie dev made a new entry in the series in the 2010s, in PICO-8, and then got the original developers' blessing! Anyway it's great.

Finally, Adam Saltsman has been making games with PICO-8 lately. He uploaded a minimal example of a Sokoban-like in just 25 lines of code which I based Snek off of, and has released several games made in PICO-8 lately. Adam is a legendary indie developer, and one of the people who made me start making games in the early 2010s. I really recommend playing his stuff, following him on social media, and learning from his approach and thinking!

PICO-8 has a really wonderful community and lots of subgroups you can find on Discord, Mastodon and other internet places.