Comp 388/488 - Game Design and Development

Spring Semester 2018 - Week 3

Dr Nick Hayward

Python and Pygame

intro

- a brief consideration of development, specifically with Python and Pygame
- install instructions for Python 3.x and Pygame
  - *Python & Pygame setup - OS X*
  - *Python & Pygame setup - Windows 10*
- Pygame is a powerful and useful set of modules to help develop and create games with Python
- best place to start is simply by visiting the website for the Pygame modules
  - *Pygame - Getting Started*

Python and Pygame - game development

game template - intro

- we may create a template file for starting our Pygame based projects
  - *a number of ways to setup a template for such game based development*
- there are a few common requirements we may start to add to a template, e.g.
  - *import required modules*
    - e.g. *pygame*, *sys*...
  - *define default settings for a Pygame window*
  - *initialise Pygame*
  - *setup the required game loop*
    - logic executed for each frame in our game...
  - *process inputs*
    - listen for events within the game
    - track events with Pygame
  - *update the game for any required changes...*
  - *rendering of the game and its graphics*
    - draw the game to the Pygame window
  - *monitor frames per second (FPS)*
    - optional for template
    - where applicable for a given game...

Python and Pygame - game template

part 1 - import

- start by importing Python modules
  - *e.g. pygame module*

```
# import modules for pygame template
import pygame
```

- we may also import the `sys` module
  - *may use as a way to exit the game*

```
# import modules for pygame template
import sys
import pygame
```

Python and Pygame - game template

part 2 - window defaults

- then add some defaults for a *window* in Pygame
  - *defining our variables as follows*

```
# variables for pygame
winWidth = 800
winHeight = 600
FPS = 30
```

- we're setting the default window size - width and height
  - *and the frames per second for the game*
- FPS may be added for applicable game types
  - *sets how fast the game will update per second on each system*
  - *may update this value for each game's requirements*
- **game loop** will then reflect the number of frames per second
  - *loop will now run 30 times per second*
  - *e.g. each loop is set to 1/30 of a second*

Python and Pygame - game template

part 3 - initialise

- then add general initialisation for our game's initial settings
  - *start by initialising Pygame, and the sound mixer*
  - *sound mixer allows us to play back sound at various points in our game*
- then create our screen or window for the game
  - *and add a brief caption for this window*
- if we're going to define the FPS for our game
  - *we also need to define a clock*
- clock helps us track how fast the game is going
  - *allows us to ensure that we're maintaining the correct FPS*

```
# initialise pygame settings and create game window
pygame.init()
pygame.mixer.init()
window = pygame.display.set_mode((winWidth, winHeight))
pygame.display.set_caption("game template")
clock = pygame.time.clock()
```

Python and Pygame - game template

part 4 - game loop

- now setup and initialised the basics for our template
  - *need to add a basic game loop to our Pygame template*
- **game loop** is one of the key requirements for developing a game
  - *including with Python and Pygame*
- *Game loop* is executed for every frame of the game
  - *three processes will happen as part of this loop*
- **processing inputs** (aka events)
  - *responding to interaction from the player with the game*
  - *e.g. keyboard press, mouse, game controller...*
  - *listening for these events, and then responding accordingly in the game's logic*
- **updating the game**
  - *updating, modifying anything in the game that needs a change*
    - e.g. graphics, music, interaction &c.
  - *a character moving - need to work out where they might be moving &c.*
  - *characters, elements in the game collide*
    - what happens when they collide? &c.
    - i.e. responding to changes in state and modifying a game...
- **rendering to the screen**
  - *drawing modifications, updates, &c. to the screen*
  - *we've worked out what needs to change*
  - *we're now drawing (rendering) those changes*
- if using FPS for game type
  - *may also need to consider how many times this game loop repeats*
  - *i.e. frames per second that this loop repeats*
  - *FPS may be important to ensure game is not running too fast or too slow*

Python and Pygame - game template

part 5 - add game loop

- we'll need to add a game loop to control and manage this pattern
  - *we're listening for inputs, events...*
  - *then updating the game*
  - *and finally rendering any changes for the user*
- we can add a standard `while` loop as a our primary game loop

```
# define boolean for active state of game
active = True
# create game loop
while active:
    # 'processing' inputs (events)
    # 'updating' the game
    # 'rendering' to the screen
```

- loop will follow defined pattern
  - *processing inputs (events)*
  - *updating the game*
  - *rendering to the screen*
- boolean `active` allows us to monitor the active state of the game loop
  - *as long as the value is set to `True` it will keep running*
  - *update this value to `False` and we may exit the game loop*
  - *we'll also see other ways of handling this exit...*

Python and Pygame - game template

part 6 - process inputs

- as the game is running
  - *a player should be able to interact with the game window*
  - *e.g. clicking the exit button, perhaps a keyboard, mouse or controller button...*
- if we consider the nature of a `while` loop
  - *we may initially see an issue with the underlying logic*
  - *e.g. the loop is either updating or rendering*
  - *what happens if a user clicks a button on the keyboard?*
- we need to be able to listen and record all events for our game
  - *regardless of the current executed point in the `while` loop*
- if not, only able to listen for events at the start of the loop
  - *as part of the processing logic*
- thankfully, Pygame has a solution for this issue

Python and Pygame - game template

part 7 - Pygame event tracking

- Pygame is able to keep track of each requested event
  - *from one executed iteration of the game loop to the next*
- it remembers each and every event
  - *as the the game's `while` loop executes the updating and rendering logic*
- as the `while` loop executes the *processing* logic
  - *we're able to check if there have been any new events*
- e.g. now add a simple `for` loop
  - *check for each and every event that Pygame has saved*

```
...
for event in pygame.event.get():
    ...
```

- start by checking for an event registered as clicking on the exit button
  - *a user request to close the current game window*

```
...
for event in pygame.event.get():
    # check for window close click
    if event.type == pygame.QUIT:
        # update boolean for running
        active = False
```

- checking for a saved event
  - *simply indicates the user wants to close the current game window*
- update the value of the boolean for the active game
  - *setting the value of the `active` variable to `False`*
  - *game loop, our `while` loop, will now exit*
- then add a call to quit Pygame at the end of our current Python file. e.g.

```
...
pygame.quit()
```

- game will now exit, and the Pygame window will close

Python and Pygame - game template

part 8 - double buffering

- as we start to render colours, lines, shapes &c. to our Pygame window
  - *need to be careful not to re-render everything for each update*
  - *if not, our game will become **very** resource intensive...*
- we can use an option known as **double buffering**
- in Pygame, this uses a concept of pre-drawing
  - *then rendering as and when the drawing is ready to be viewed by the player*
  - *drawing is flipped to show the rendering to the player*
  - *e.g. we can the following to our template*

```
...
# flip our display to show the completed drawing
pygame.display.flip()
```

- **n.b.** `flip` must be the last call after drawing
  - *if not, nothing will be displayed to the game's player*

Python and Pygame - game template

part 9 - monitor FPS

- *game loop* may also need to monitor and maintain setting for our game's FPS
- currently FPS set to 30 frames per second
- within the logic of our *game loop*

```
...
# check game loop is active
while active:
    # monitor fps and keep game running at set speed
    clock.tick(FPS)
...
```

- Pygame is now able to keep our game running at the defined frames per second
- as the loop runs
  - *it will always ensure that the loop executes the required 1/30 second*
- as long as the loop is able to *process*, *update*, and *render*
  - *within this defined time period of 30 fps, rendering will be smooth*
  - *if not, usually the update is taking too long*
  - *our game will run with lag, appear jittery to the player*
  - *may need to consider optimisation for code and logic...*

Python and Pygame - game template

part 10 - finish the template

- as we're only listening for the exit event on the game window
  - *we don't currently have any game content to update*
- our current template has set up a game window, and environment
  - *to test initial setup and initialisation*
  - *then allow a player to exit the game and window*
  - *e.g.*

```
...
# quit the Pygame window, exiting the game
pygame.quit()
...
```

Python and Pygame - game template

part 11 - another example template

```
# import modules for pygame template
import pygame, sys

# variables for pygame
winWidth = 800
winHeight = 600

# variables for commonly used colours
BLUE = (0, 0, 255)

# initialise pygame settings and create game window
pygame.init()
window = pygame.display.set_mode((winWidth, winHeight))
pygame.display.set_caption("game template")

# define game quit and program exit
def gameExit():
    pygame.quit()
    sys.exit()

# create game loop
while True:
    # 'processing' inputs (events)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            gameExit()
    # 'updating' the game

    # 'rendering' to the window
    window.fill(BLUE)
    # 'flip' display - always after drawing...
    pygame.display.flip()
```

Games and playtesting

player experience goals and aims - part 1

- need to consider goals for a player's experience with our game
- commonly known as **player experience goals**
  - *goals that we may define for a player whilst testing and playing our game*
  - *not defined features of the game (specific gameplay, mechanics &c.)*
  - *consider them descriptions of interesting, useful, unique situations or scenarios*
- for example:
  - *a player may progress through a particular level*

> *a player should begin rapidly, and encounter a sense of frustration as they tackle sets of problems. As they progress from problem to problem, this frustration is replaced with a sense of achievement. Ultimately, satisfaction results as they complete the level.*

- another common example is a description of structure for a particular gaming experience, e.g.

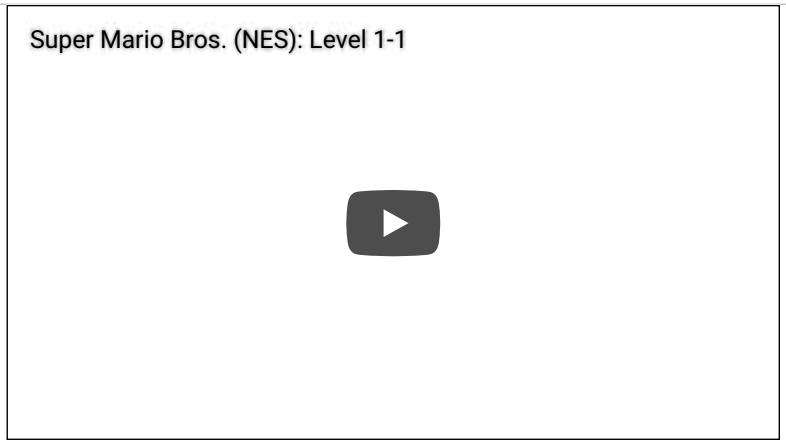
> *a player should be free to wander and experience the game at their own pace, and in their chosen order...*

Games and playtesting

player experience goals and aims - part 2

- we're trying to describe our game from the perspective of a player
  - *not as a designer and developer*
  - *e.g. what should a player expect from aspects of the game...*
- such goals also prove very useful for initial game planning
  - *plan initial design and layout...*
  - *helps prevent an initial focus on the minutiae of a game's development*
- instead, plan the game as a player
- may also use such goals later in each playtesting scenario
  - *helps correlate expected game design with playtesters' expectations*

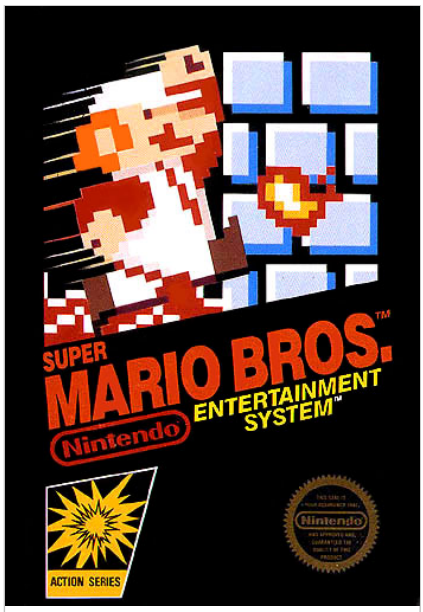
Video - Super Mario Bros. - Level 1



Source - YouTube

Games and playtesting

Quick exercise



Nintendo - Super Mario Bros.

- consider player experience goals for the first level of Super Mario Bros.
- outline your top three player goals for this level...
- outline your top three designer goals for this level...

Games and playtesting

initial prototypes and playtests

- prototype and test our initial game concepts
- not necessarily digital, interactive prototype
- simply a playable version of the initial game idea
- may start with a physical prototype of game's
  - *core concepts*
  - *playable mechanics*
  - *structure*
- physical prototype is a useful option
  - *perceive, test, and demonstrate core concepts*
  - *useful before starting coding and development*
- physical prototype may use different mediums, e.g.
  - *pen, paper, cards, cardboard...*
  - *perhaps even act out parts of the game...*
- this technique helps in many respects, e.g.
  - *perfect, as far as possible, initial game model*
  - *then pass model to artists, developers, producers...*
- we're checking player experience goals
  - *ensure playtesters may achieve these goals...*

Video - Paper Prototyping

example paper prototype - initial concept 1

Source - YouTube

Video - Prototyping by Acting

Walt Disney

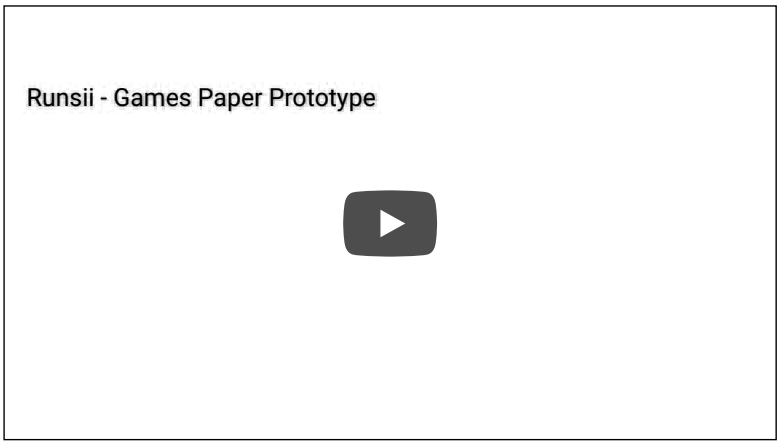
Games and playtesting

design and development patterns - part 1

- consider general ideas and concepts for your game project
  - *discuss, read, watch, listen...anything to help inspire ideas and concepts*
  - *set player experience goals for the type of game you'd like to create*
  - *consider concepts and mechanics you want in your game*
  - *brainstorm initial top 3-5 ideas in your project group*
- prototype - stage 1
  - *create an initial physical prototype for your top 3 ideas (where applicable)*
  - *useful to help with selling your game concept (e.g. to funders, other developers, testers...)*
  - *example artwork, character concepts, story themes and outlines...*
  - *act out gameplay examples...*
- prototype - stage 2
  - *start creating initial gameplay digital prototypes*
  - *interactive examples to test core gameplay*
  - *several prototypes will usually be created*
  - *each testing different concepts and examples for your game*
  - *try to keep this quick, and easy to modify and update*
  - *do not get too preoccupied with the overall fidelity...*
  - *playtest these digital prototypes*

Video - Paper Prototyping

example paper prototype - detailed concept 1



Source - YouTube

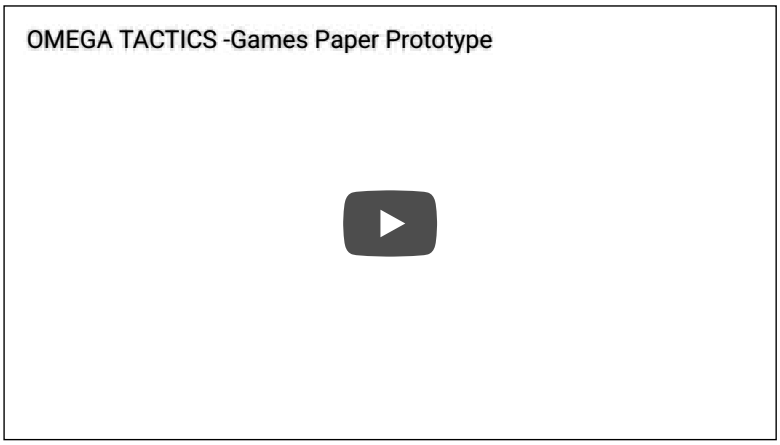
Games and playtesting

design and development patterns - part 2

- document design and development requirements
  - *use any notes, sketches, lists, &c. created during previous steps*
  - *these will help suggest structure and ideas for formal documentation*
  - *compile a full list of requirements, and development goals for your **actual** game*
  - *try to keep this documentation open to collaborative usage and editing*
  - *it will need to adapt and update as you develop the game*
- build and produce your game
  - *check each team member knows exactly what they need to do...*
  - *consider desired milestones for your game's development*
  - *check game design and development at each milestone*
  - *evaluate current state of game as a group*
  - *start developing the final game...*
- test, test, and test again
  - *after you reach a given milestone, quality assurance is now possible*
  - *should highlight working, well considered gameplay...*
  - *it will not resolve all issues*
  - *playtesting may continue to ensure quality and accessibility for players*

Video - Paper Prototyping

example paper prototype - detailed concept 2



Source - YouTube

Games and playtesting

benefits and usage

- this may seem a lot of work and preparation
  - *before reaching digital design and development*
- a few options to customise iterative patterns
  - *industry provides a few examples*
- as with most guidelines, recommendations, and systems
  - *modify them to fit your game's specific requirements*
- e.g. physical prototypes
  - *may be less useful and applicable for well established mechanics and gameplay*
- industry game projects will often skip this step
  - *may not incorporate as part of iterative design and development process*
  - *assuming game uses existing or well established mechanics and gameplay*
- many companies produce games as expansions, updates to existing titles
  - *variations on standard, well tested game mechanics...*
  - *designers and developers have a good idea how the game will work*
  - *they feel comfortable skipping ahead in the process*
- may also be due to industry pressures in general
  - *e.g. costs, timescales, resources, player perceptions...*

Games and playtesting

industry example - part 1

- such initial steps, including physical prototypes, become crucial
  - *e.g. if we are designing innovative mechanics and gameplay*
- for new examples and concepts
  - *crucial to plan and test thoroughly*
- Electronic Arts (EA) has used such processes
- EA introduced internal training for pre-production methods in the mid-2000s
  - *e.g. workshops on physical prototyping and playtesting*
- Jeremy Townsend, who has worked at EAs Tiburon studio
  - *best known for the Madden and Tiger Woods series of games*
  - *has used such **rapid prototyping** and pre-production methods*
  - *used these methods to help inform game development*

> *"Stay away from 3D prototyping if at all possible. Most game problems can be solved in 2D, even on paper," he said. "The Play's the thing - think of 3D prototyping as a big gun, you only want to use it as a last resort."*

develop - EA at Grand Rapids

industry example - part 2

- EA has also used Microsoft's XNA development tools
  - *e.g. for the XBox 360 console and Windows PCs*
  - *helps develop ideas quickly and efficiently*
- rapid prototyping still plays a key role for EA

> "*if something doesn't work you can correct away from it*"

develop - EA at Grand Rapids

- **Spore**, for example, was released by EA in 2008
  - *example of a god game*
  - *well-known for its use of procedural generation*
  - *used this type of pre-production testing and development*
  - *included the creation of many different prototypes*
  - *e.g. Spore - Prototypes*

Image - EA *Spore*

EA - Spore, 2008

Image - Game Jams

Global Game Jam

Games and playtesting

industry example - part 3

- Global Game Jam](http://globalgamejam.org/)
  - *designers, developers, &c. from around the world...*
  - *20th to 22nd January 2017 in Hawaii*
  - *more than 36,000 participants in 702 sites...*
  - *more than 7,000 games created*
- this year's theme was **waves**
- diversifiers available as well
  - *optional constraints...*

example diversifiers

- **Spaced**
  - *only play using the spacebar - no mouse, other inputs*
  - *testing accessibility, design*
- **Old Masters**
  - *game's art style inspired by a master artist's style*
  - *e.g. Picasso, Klimt, Van Gogh*
  - *testing art*
- **Chipping In**
  - *game may use only 8 bit style audio, visuals, or both*
  - *testing audio, art*
- many more...

Playtesting fun

Super Mario Bros.

Play a fun copy online,

- Super Mario Bros. emulator copy

Games and formal structure

intro

- start to design and build our games
  - *consider components and structures that make a game*
  - *something that people will actually want to play*
- different interpretation of the nature of a game
  - *underlying premise is reinforced by particular structures*

Image - Draughts vs Space Invaders

pick a game

Draughts/Checkers	Space Invaders
	

References

- develop. *EA at Grand Rapids*. http://www.develop-online.net/tools-and-tech/grand-rapids/0116020. 2007.
- David, S. *Game Over: How Nintendo Conquered the World*. Vintage Books. New York. 1994. P.51.
- Electronic Arts. Spore Prototypes. http://www.spore.com/comm/prototypes. 2008.
- Global Game Jam
- Wikipedia
- God Game
- Shigeru Miyamoto
- Spore - 2008

Videos

- Super Mario Bros. - Level 1
- Paper Prototyping
- initial concept 1
- detailed concept 1
- detailed concept 2