Comp 388/488 - Introduction to Game Design and Development

Spring Semester 2017 - Week 4

Dr Nick Hayward

Contents

- Games and formal structure
- Games and planning
  - *incl. a few game examples*
- Python and Pygame
  - *intro*
  - *templates*

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
	

Games and formal structure

structures

- regardless of the specifics of each game
  - *analogue vs digital*
  - *perhaps commercial compared to open source*
  - *turn-based vs a shooter game*
  - *...*
- still perceive each example as a game
  - *something that people will want to play*
- obvious disparities between **Draughts** and **Space Invaders**
  - *may identify similarities in general experiences of both games*
  - *sufficient to evolve a definition of a game*
- each game shares a few similarities and traits that inherently make a *game*, e.g.
  - *players*
- objectives
- procedures & rules
  - *including implied boundaries*
- conflict, challenge, battle...
- outcome, end result...

Games and formal structure

players - part 1

- players are an obvious similarity
  - *but one that still helps to define our games*
- each game requires players
  - *a description of each game defines an experience structured for its players*
  - *we're defining the game based upon interactive participation*
- gameplay scenarios may be different for each game
  - *unifying factor is the concept of player participation in the game experience*
  - *each player is an active contributor to the respective game*
  - *they make decisions, adopt roles, become invested in the gameplay...*

Games and formal structure

players - part 2

- to play each game as defined
  - *a player must voluntarily accept the defined rules and structures for the game*
- initially defined by Bernard Suits as a **lusory attitude**
  - *he considered rules and games as,*

> *To play a game is to attempt to achieve a specific state of affairs...where the rules are accepted just because they make possible such activity.*

Suits, B. *The Grasshopper: Games, Life and Utopia*. Broadview Press. 3rd Edition. 2014.

- the **lusory attitude** becomes an inherent requirement for each player
  - *an acceptance of arbitrary rules for each game to permit gameplay*
  - *forms a key part of the player's required emotional and psychological states*
- how we manipulate, coerce such states will often be key to the success of our gameplay
- need to be careful how far we push or skew such rules within our game
  - *too far - player may snap, and reject the game*
  - *game may be perceived as too difficult, demeaning, removed from experiential reality...*

Games and formal structure

objectives

- each game clearly defines goals and requirements for play and players
  - *in effect, aspirations for the game...*
- in *Draughts*, each player is trying to ensure their opponent
  - *either loses all of their pieces*
  - *or can no longer move any of the remaining pieces*
- in *Space Invaders*, a player is trying
  - *to defeat rows of aliens (often five rows of eleven aliens)*
  - *whilst preserving their own defensive bunkers and lives*
- both games offer different overall objectives, but they feature
  - *interactive objectives to reach a defined conclusion*
- compare this to a passive act such as
  - *listening to music, reading a book, or watching a movie*
- each game's objective becomes a trait
  - *a requirement for the game itself*
- if not, we're simply watching
  - *an inanimate board*
  - *or aliens advancing down a screen*

Games and planning

flowcharts - intro

- may create a flowchart to help outline initial gameplay
- chart acts as our first consideration of available paths within our game
  - *both successful and unsuccessful*
- we may then use this flowchart as a simple kernel for gameplay
  - *chart is then developed and enhanced as we expand our game*
- a flowchart is a simple concept
- it allows us to create a representational diagram
  - *of pathways or flow for a given series of steps that form a process*
  - *process may be part of a task*
  - *which we may then combine to allow completion of a goal...*

Games and planning

flowcharts - design

- we may design and create our flowchart using any number of shapes and connecting paths
  - *often represented as directional lines*
  - *shapes will normally represent an action or task that a player may complete*
- we can also add conditional options to the flowchart
  - *may represent choices a player may make*
  - *within the logic of the game, and its gameplay*
- for example, we may consider the following outline
  - ***Enter the Mummy's Tomb*** *- a basic text-based game*
  - *a player is in a fantasy world based on Ancient Egypt*
  - *our player is exploring the Valley of the Kings*
  - *each tomb contains either a Pharaoh's burial treasure or a Mummy*
  - *a Pharaoh's mummy does not like being disturbed*
  - *the player approaches the entrance to a tomb*
  - *they must choose whether to enter or not*

Games and planning

outline and structure - *Enter the Mummy's Tomb*

- basic logic for this game may use the following outline and structure
- a Python based game, *Enter the Mummy's Tomb*
  - *import statements*
  - *import modules* `random` *and* `time`
  - *define functions for app structure and logic*
  - *output the intro to the game*
  - *allow a user to choose a cave*
  - *check chosen cave*
  - *simple option to play the game again*
  - *while loop for game play option (yes or no)*

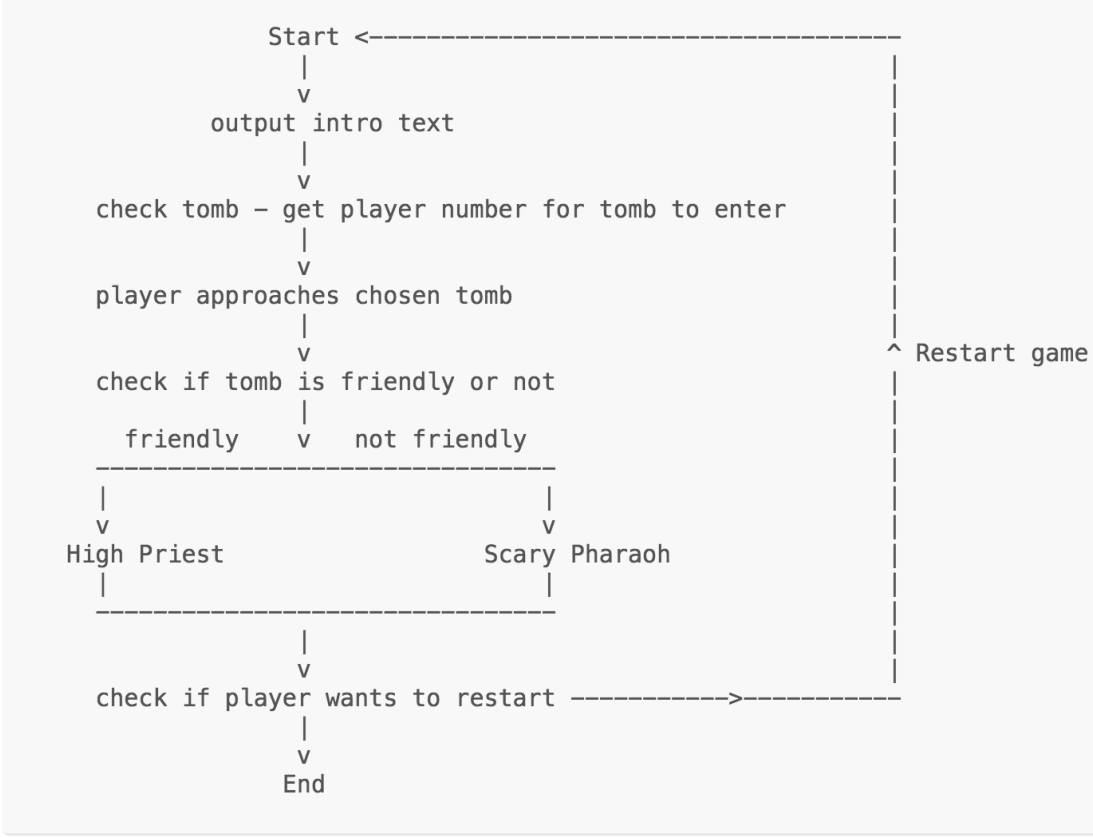
Games and planning

flowcharts - *Enter the Mummy's Tomb*

- to start designing our game
  - *we need to consider the path and options our player may choose*
- i.e. how they may progress from start to finish for such games
- our game follows the pattern of a *text adventure*
  - *a type of interactive fiction game*
  - *an example similar to the famous Zork game*
- may often depict the structure and options using a visualisation
  - *a flowchart is a good example for this type of game and logic*

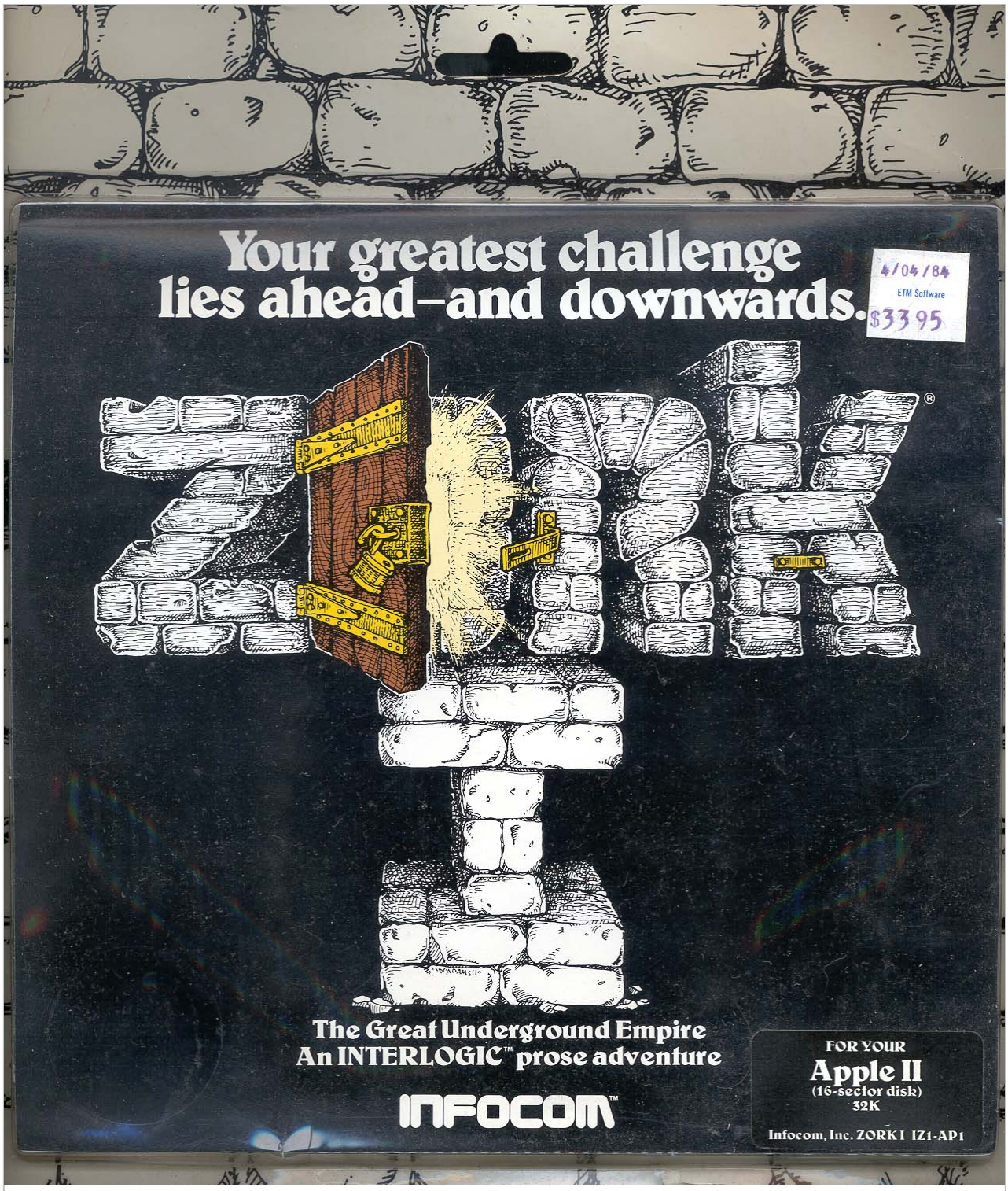
Image - Flowchart - Example 1

Enter the Mummy's Tomb



Flowchart - Enter the Mummy's Tomb

Image - Zork



Zork

Games and planning

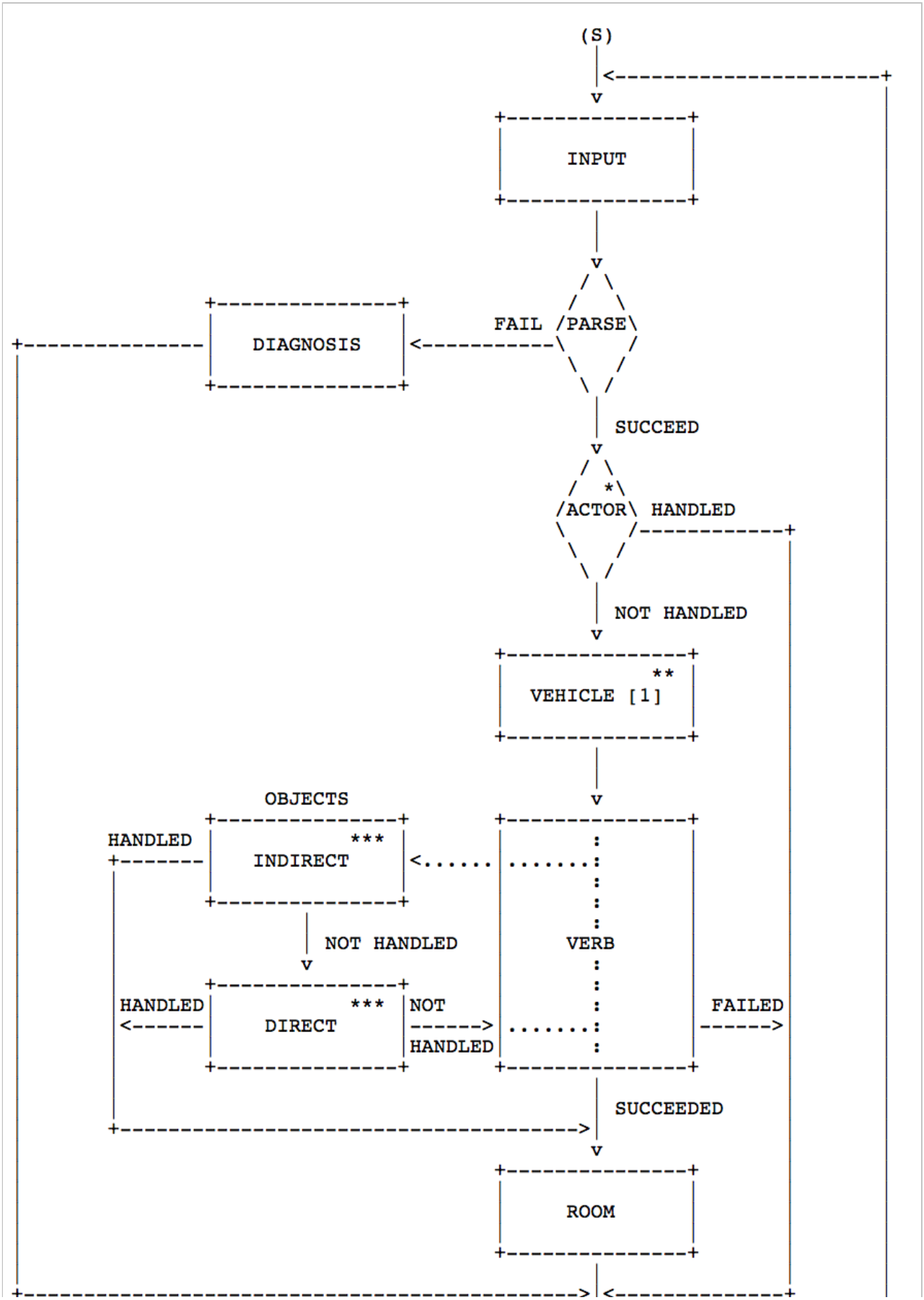
Zork

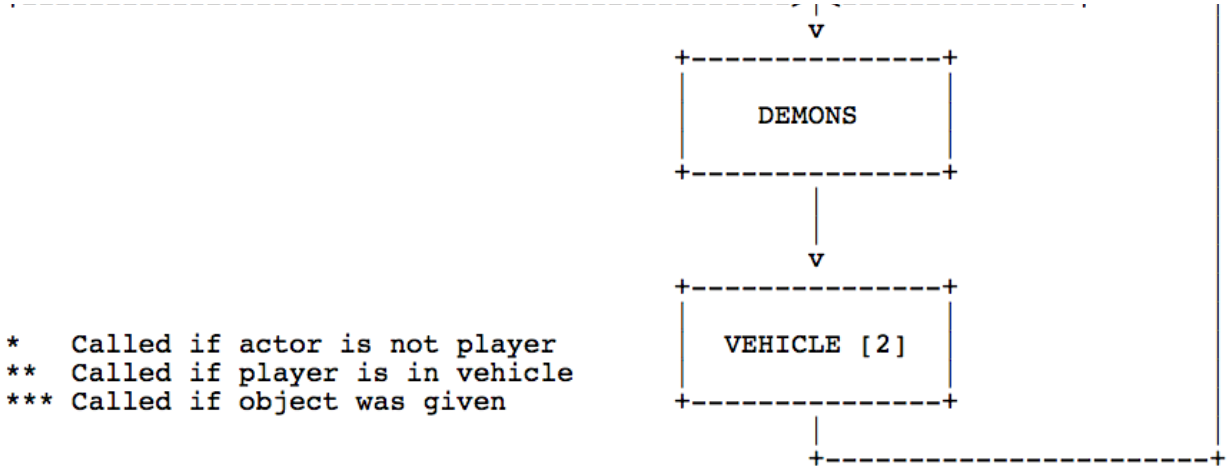
- **Zork**, one of the best known text-based adventure games
  - *written in 1977 for the PDP-10 mainframe computer*
  - *second text-based adventure game ever written - first was Colossal Cave Adventure*
    - written in 1976 for the PDP-10
  - *both games were interactive fiction*
  - *set in the ruins of an ancient empire lying far underground*
- player's character is simply an anonymous adventurer
  - *who is venturing into this dangerous land in search of wealth and adventure*
- primary goal of this game is to return alive
  - *from exploring the "Great Underground Empire"*
- a victorious player will earn the title of *Dungeon Master*
- game's dungeons include a variety of objects...
  - *interesting and unusual creatures, objects, and locations*
- best known creature is the ferocious but light-fearing *grue*
  - *a term for a fictional predatory monster that dwells in the dark*
- ultimate goal of Zork I is to collect the Twenty Treasures of Zork
  - *and install them in the trophy case*
- finding the treasures requires solving a variety of puzzles
  - *such as the navigation of two complex mazes*
- end of Zork I becomes the entrance, and beginning to the world of Zork II
- fantastic text-based game
  - *feels part fantasy, part classical mythology, and part sci-fi...*

- Download the Zork games for Mac and Dos/Windows at the following URL,
  - *Infocom - Zork*

Image - Flowchart - Example 2

Zork

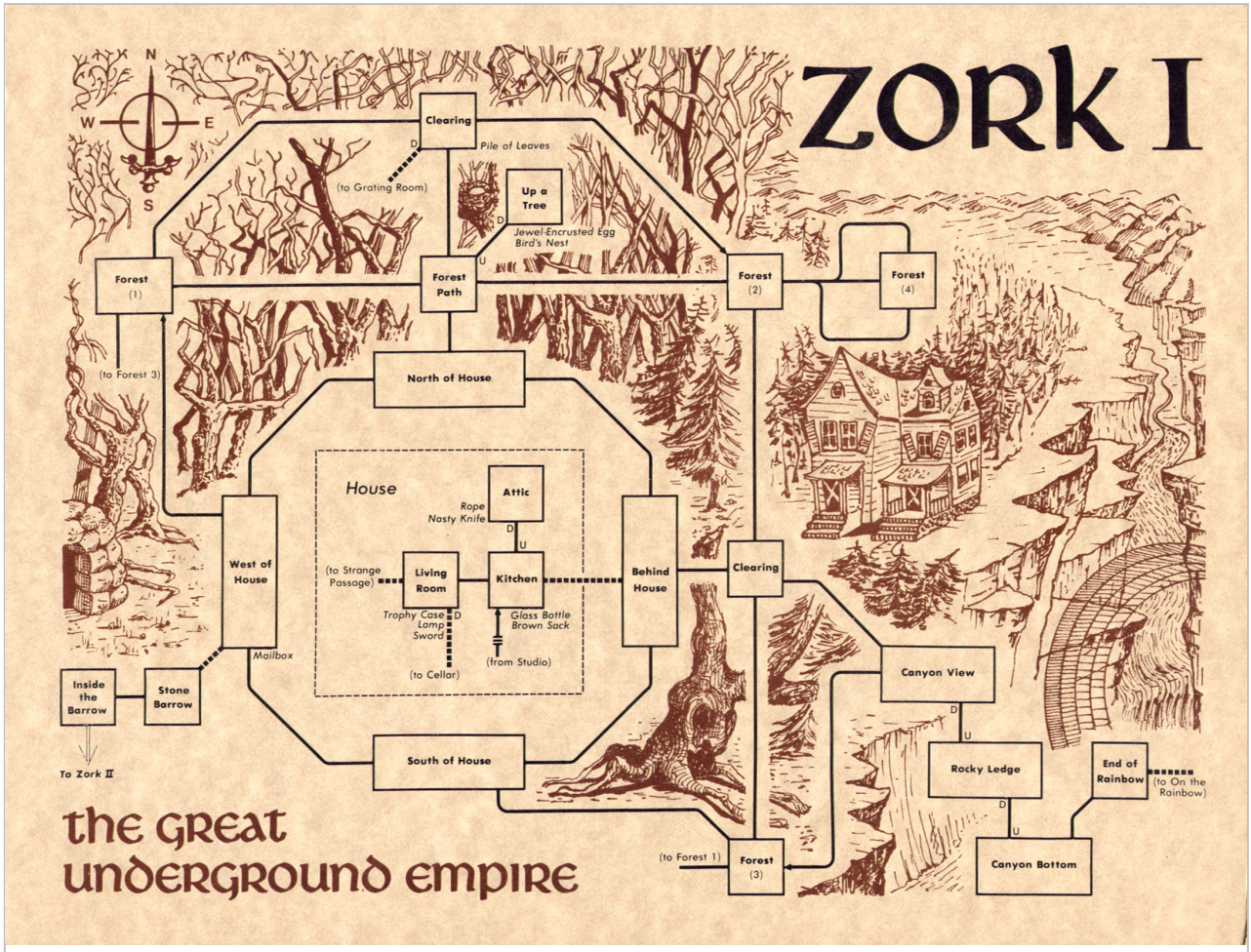




Flowchart - Zork - Logic

Image - Flowchart - Example 3

Zork Map



Flowchart - Zork - Map

Games and planning

quick exercise

Briefly describe your basic game objectives for the following game ideas.

Then, briefly draw an outline flowchart for this game to allow a player to play from the start to the end of an example objective.

Game ideas include:

- **a single player in a locked square room**
  - *each of the four doors may be opened by solving a series of puzzles, challenges, or mini-games within the room*
  - *the room decreases in size as time progresses in the game*
- **a single player on an alien planet**
  - *the heat starts to rise as time progresses in the game*
  - *as the character's temperature rises, it starts to shrink by a proportionate amount*

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

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

- Zork - Downloads
  - *Zork - original version for PDP*
  - *Zork I - Apple 2e version*
  - *Zork I walkthrough - very useful*

References

- Suits, B. *The Grasshopper: Games, Life and Utopia*. Broadview Press. 3rd Edition. 2014.
- Wikipedia
  - *Draughts*
  - *Space Invaders*
  - *Zork*