Comp 388/488 - Game Design and Development

Spring Semester 2018 - Week 4

Dr Nick Hayward

Python and Pygame

Extra notes

- extra notes on course website & GitHub account,
  - *course website - notes*
  - *course GitHub account*
- Python and Pygame setup
  - *drawing*
  - *basic intro*
    - moving shapes
    - colours
  - *Python and Pygame*
  - *getting started*
    - animation and control of colour
    - events and user interaction
    - move and control items

Video - Interaction Events

keyboard - control shape - left to right

Video - Interaction Events

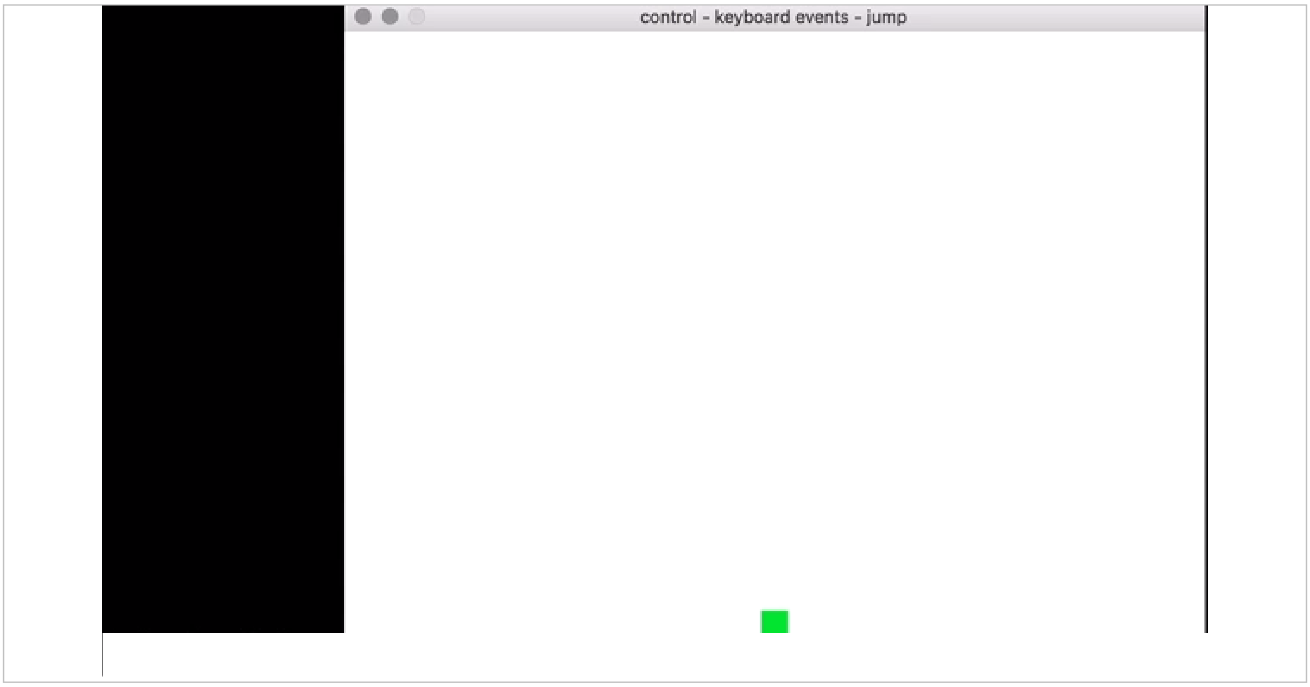
keyboard - control shape - up and down

Video - Interaction Events

keyboard - control shape - 8-point move

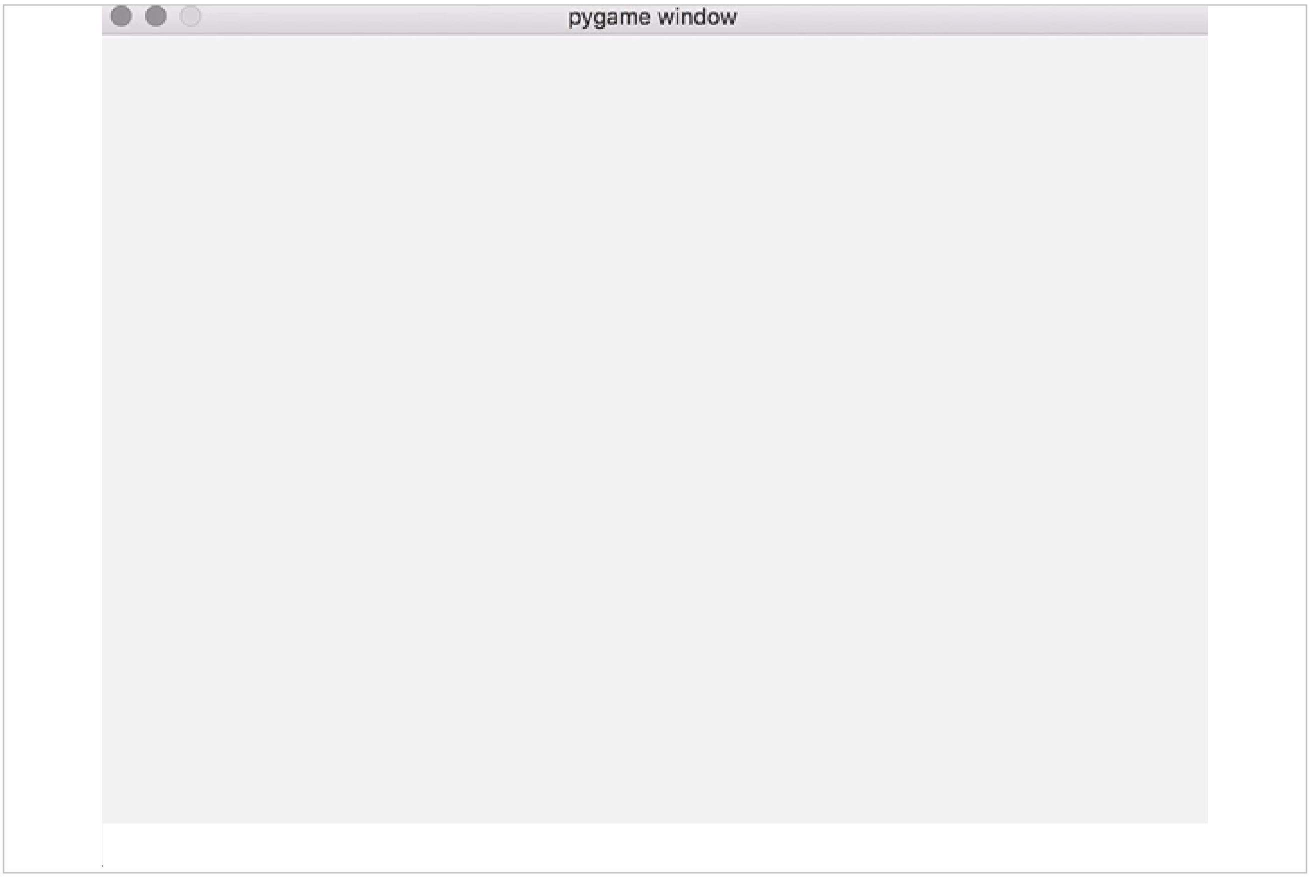
Video - Interaction Events

keyboard - control shape - jump, jump, jump...



Video - Interaction Events

keyboard - control shape - jump and freeze



Python and Pygame - events

keyboard - control shape - jump and fall

- we could make the shape move down the window
  - *e.g. by listening for an explicit key press on the* ***DOWN*** *directional key*
- it's more natural, and expected behaviour, to allow our shape to fall
  - *after the player has pressed the* ***UP*** *directional key*
  - *allowing our shape to jump, and then fall*
  - *fall with a real-world behaviour of* ***gravity***
- to make it fall, we need to check that the shape is *in the air*
- then gradually modify gravity to lower the shape
  - *lower to the original starting position in the Pygame window*

Python and Pygame - events

keyboard - control shape - jump and fall

code example

```python
def jump():
    global shapeY, shapeJY, shapeJump, gravity
    # check upward speed > 1.0
    if shapeJY > 1.0:
        # gradually decrease upward speed to less than 1.0
        shapeJY = shapeJY * 0.9
    else:
        # less than 1.0, reset to 0.0 to allow shape to fall
        shapeJY = 0.0
        # stop jump
        shapeJump = False

    # check if shape in air - use gravity to descend
    if shapeY < winHeight - shapeSize:
        shapeY += gravity
        gravity = gravity * 1.1
    else:
        shapeY = winHeight - shapeSize
        gravity = 1.0

    shapeY -= shapeJY
```

Python and Pygame - events

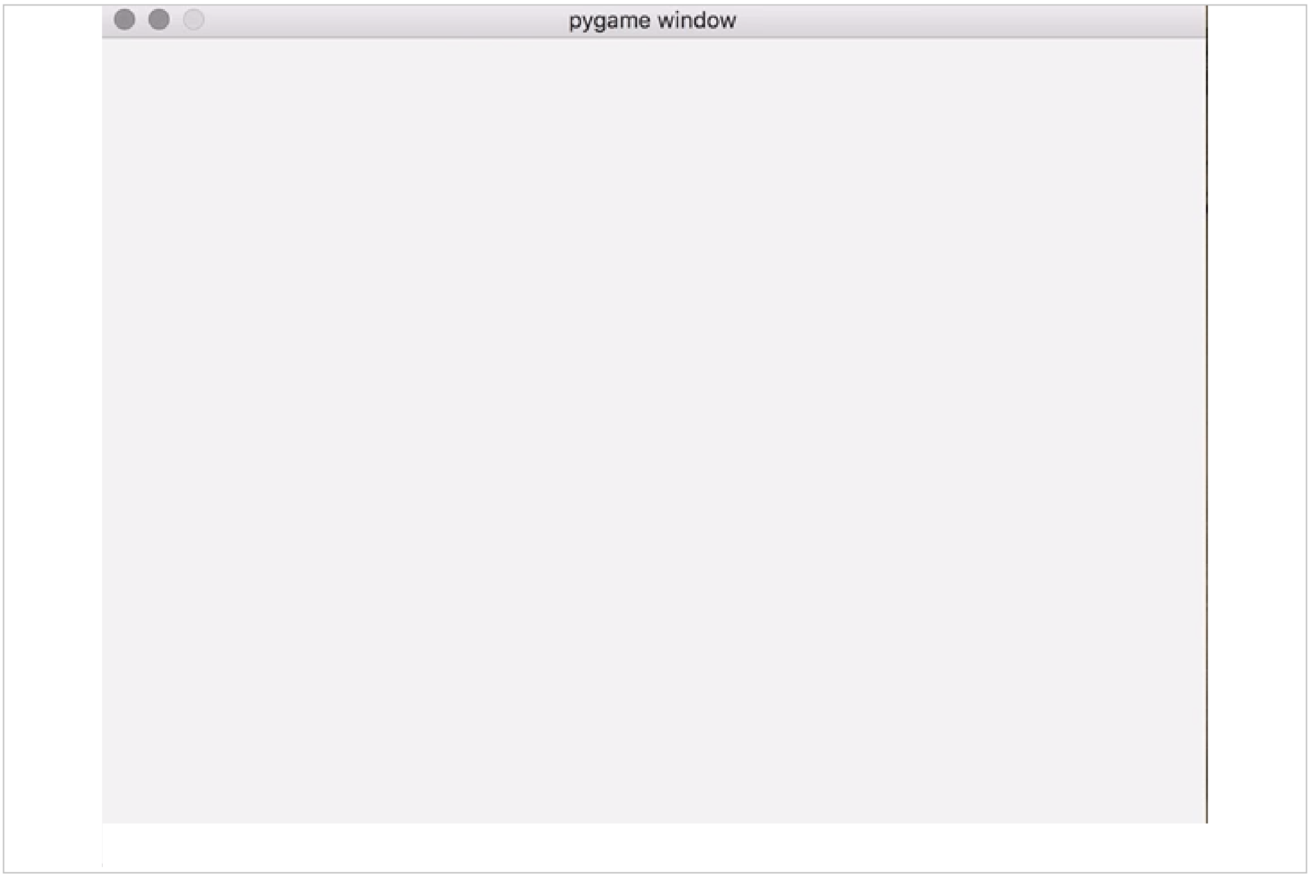
keyboard - control shape - jump and fall

code example outline

- in the previous code example
  - *start by checking whether the shape is still moving up the screen*
  - *effectively if the jump is still in progress*
- whilst the upward speed of the shape is still above `1.0`
  - *gradually start to decrease the speed*
  - *it will eventually reach a limit for the jump*
- faster we decrease this upward motion
  - *the shorter the shape will appear to jump*
- also negates the overall effect of the value of the variable `jumpHeight`
  - *now has less iterations of the game loop to move the shape up the screen*
- need to check if the shape is actually moving up the screen
  - *or effectively in the **air** for the jump*
- if not, then the shape will simply come to a halt as it rises up the screen
  - *due to the decrease in upward speed and motion*
- we need to add the perception of **gravity** to the shape's motion
  - *whilst the shape appears to be in the **air**, or jumping up the screen*
  - *start to add the number of pixels we define for the variable **gravity***
  - *add pixels to our shape's upward movement*
- as the shape starts to fall down the game window
  - *slowly increase the value of the `gravity` variable*
  - *helps to suggest a realistic downward fall*
- if not, the jump and fall will not be timed correctly
  - *a player will perceive the shape's fall as very slow*
  - *the fall will seem unrealistic, as though the gravity is too low...*

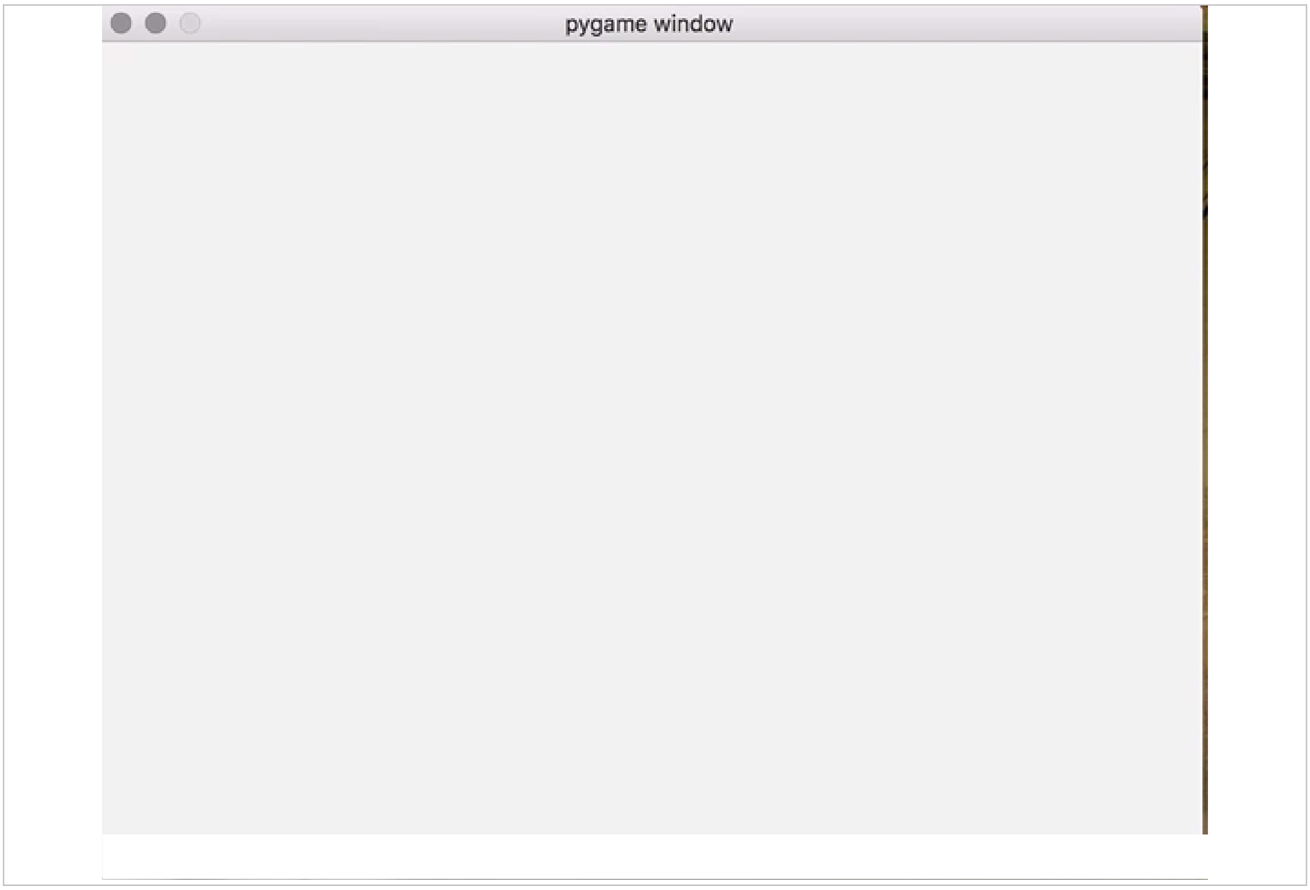
Video - Interaction Events

keyboard - control shape - jump and fall slowly



Video - Interaction Events

keyboard - control shape - jump and fall with gravity



Python and Pygame - events

keyboard - control shape - move, jump...

- update the **game loop** to include required listeners and handlers for horizontal movement
  - *add the required listener for* *`KEYUP`*
    - stop our shape from continuously moving right or left

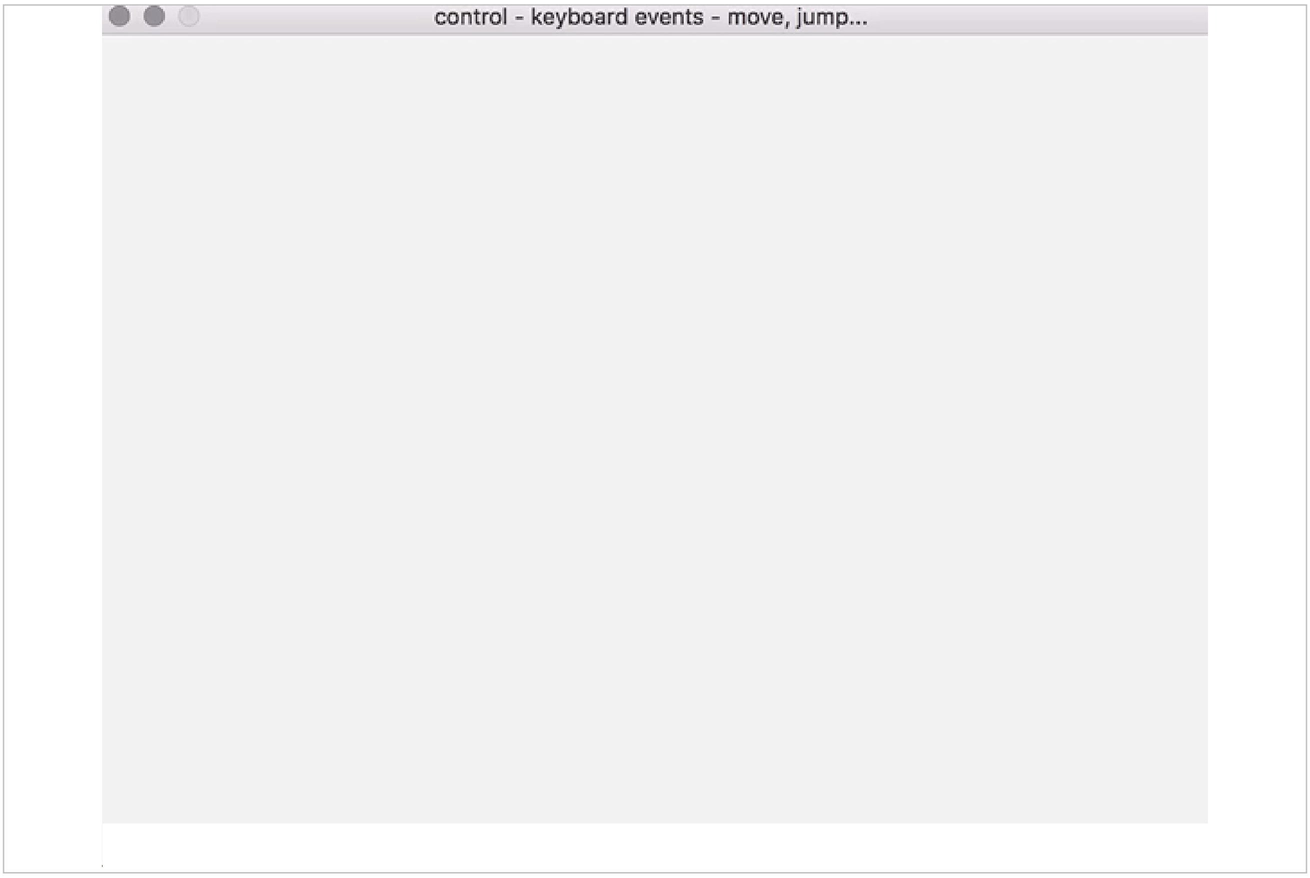
- shape can now walk and jump across the game window

```
# create game loop
while True:
    # set clock
    #msElapsed = clock.tick(max_fps)
    #print(msElapsed)
    # 'processing' inputs (events)
    for event in EVENTS.get():
        # check keyboard events - keydown
        if event.type == pygame.KEYDOWN:
            # check for directional - LEFT and RIGHT
            if event.key == pygame.K_LEFT:
                leftDown = True
            if event.key == pygame.K_RIGHT:
                rightDown = True
            # check for directional - UP
            if event.key == pygame.K_UP:
                if not shapeJump:
                    shapeJump = True
                    shapeJY += jumpHeight
            # check for ESCAPE key
            if event.key == pygame.K_ESCAPE:
                gameExit()

        # check keyboard events - keyup
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_LEFT:
                leftDown = False
            if event.key == pygame.K_RIGHT:
                rightDown = False
```

Video - Interaction Events

keyboard - control shape - move and jump



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*

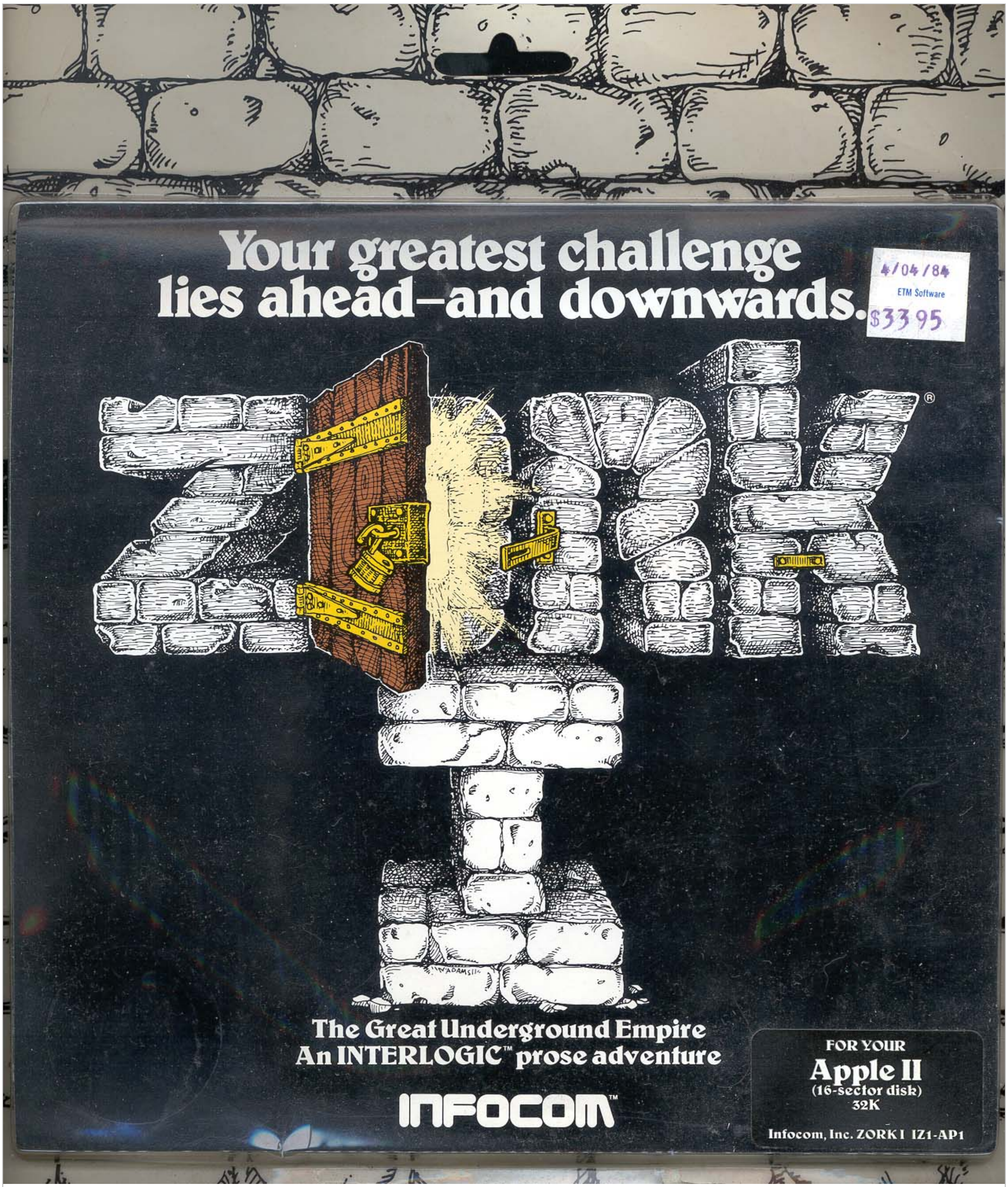
Image - Flowchart - Example 1

Enter the Mummy's Tomb

```
                  Start <-------------------------------------
                    |                                        |
                    v                                        |
              output intro text                              |
                    |                                        |
                    v                                        |
      check tomb - get player number for tomb to enter       |
                    |                                        |
                    v                                        |
      player approaches chosen tomb                          |
                    |                                        |
                    v                                        ^ Restart game
      check if tomb is friendly or not                       |
                    |                                        |
        friendly    v   not friendly                         |
      --------------------------------                       |
      |                              |                       |
      v                              v                       |
    High Priest                  Scary Pharaoh               |
      |                              |                       |
      --------------------------------                       |
                    |                                        |
                    v                                        |
      check if player wants to restart ----------->-----------
                    |
                    v
                   End
```

Flowchart - Enter the Mummy's Tomb

Image - Zork


Your greatest challenge lies ahead–and downwards.
4/04/84
ETM Software
$33 95
ZORK®
N. ADAMS
The Great Underground Empire
An INTERLOGIC™ prose adventure
INFOCOM™
FOR YOUR
Apple II
(16-sector disk)
32K
Infocom, Inc. ZORK I IZ1-AP1

Zork

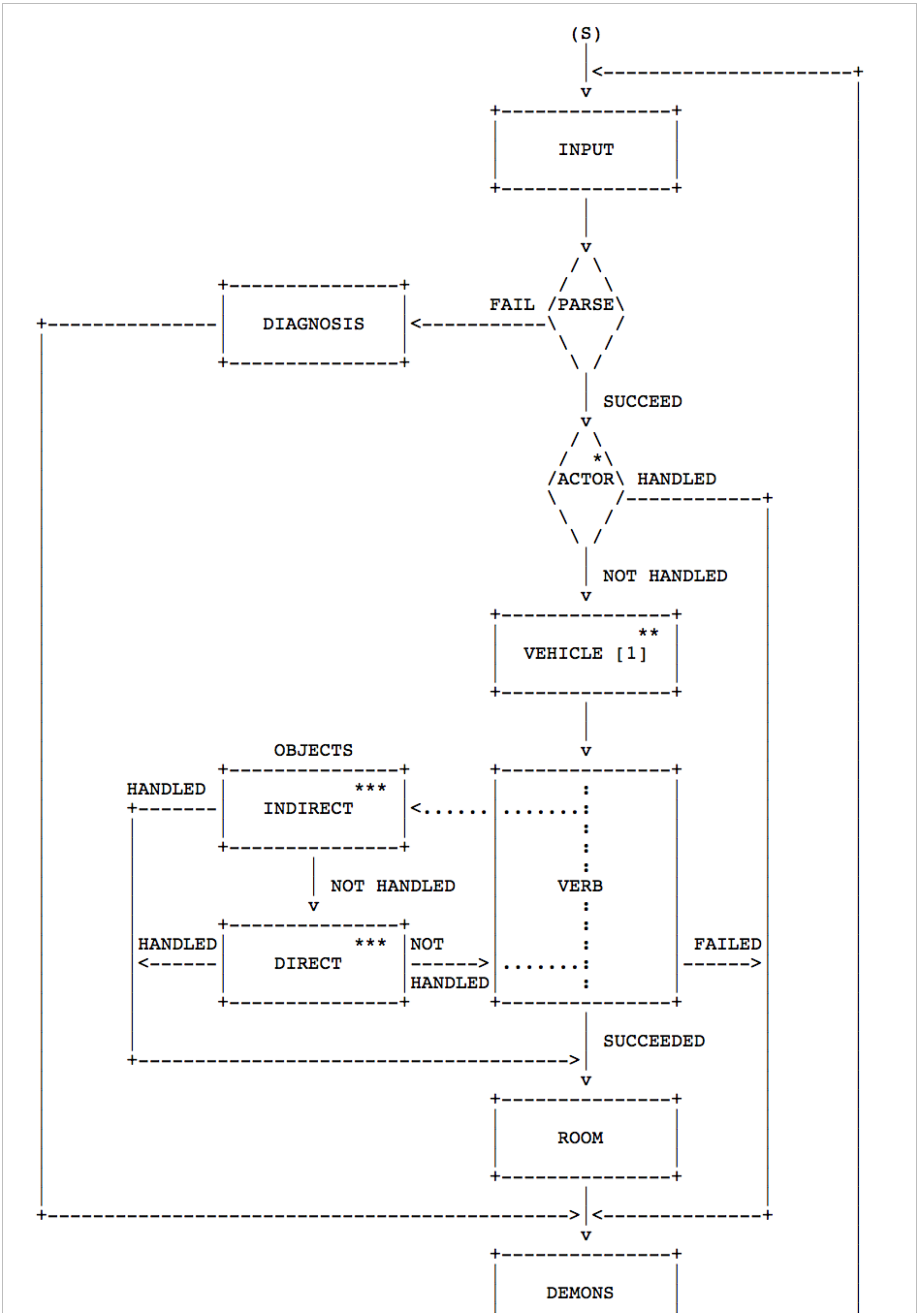
Games and planning

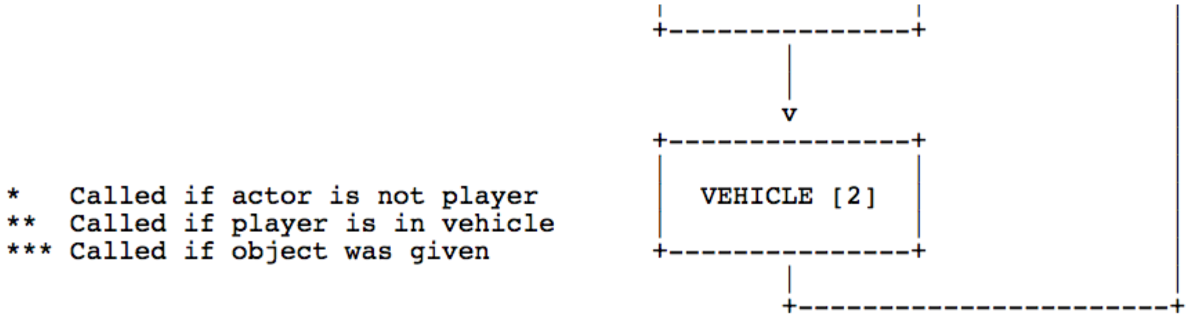
Zork

- **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



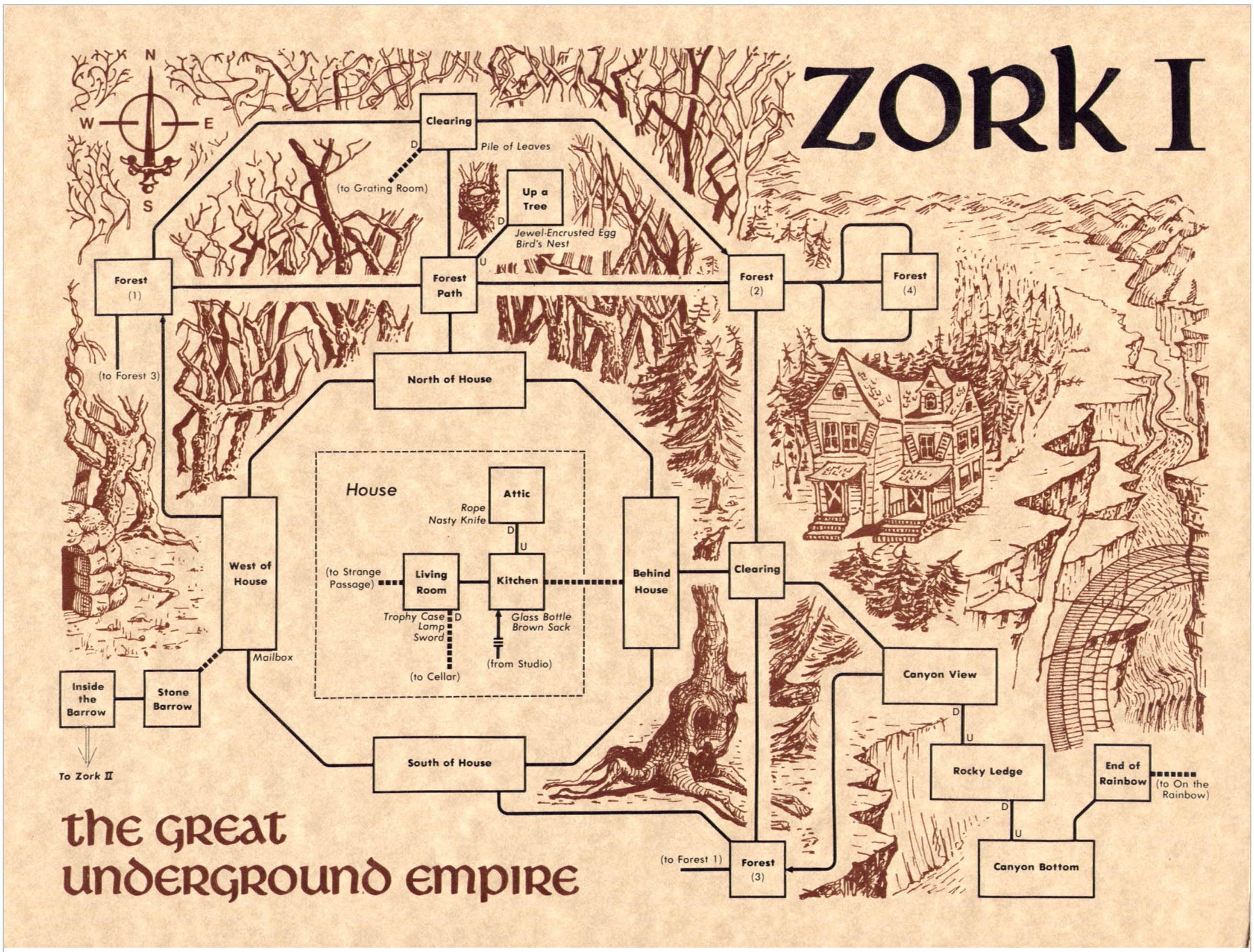


```
*   Called if actor is not player
**  Called if player is in vehicle
*** Called if object was given
```

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 - Sprites

intro

Please consult the extra notes on Pygame Sprites,

- sprites - intro

resources

- notes = sprites-intro.pdf
- code = basicsprites1.py

Python and Pygame - Sprites

image import

Please consult the extra notes on Pygame Sprites,

- sprites - set image

resources

- notes = sprites-set-image.pdf
- code = basicsprites2.py

Image - Image Sprite

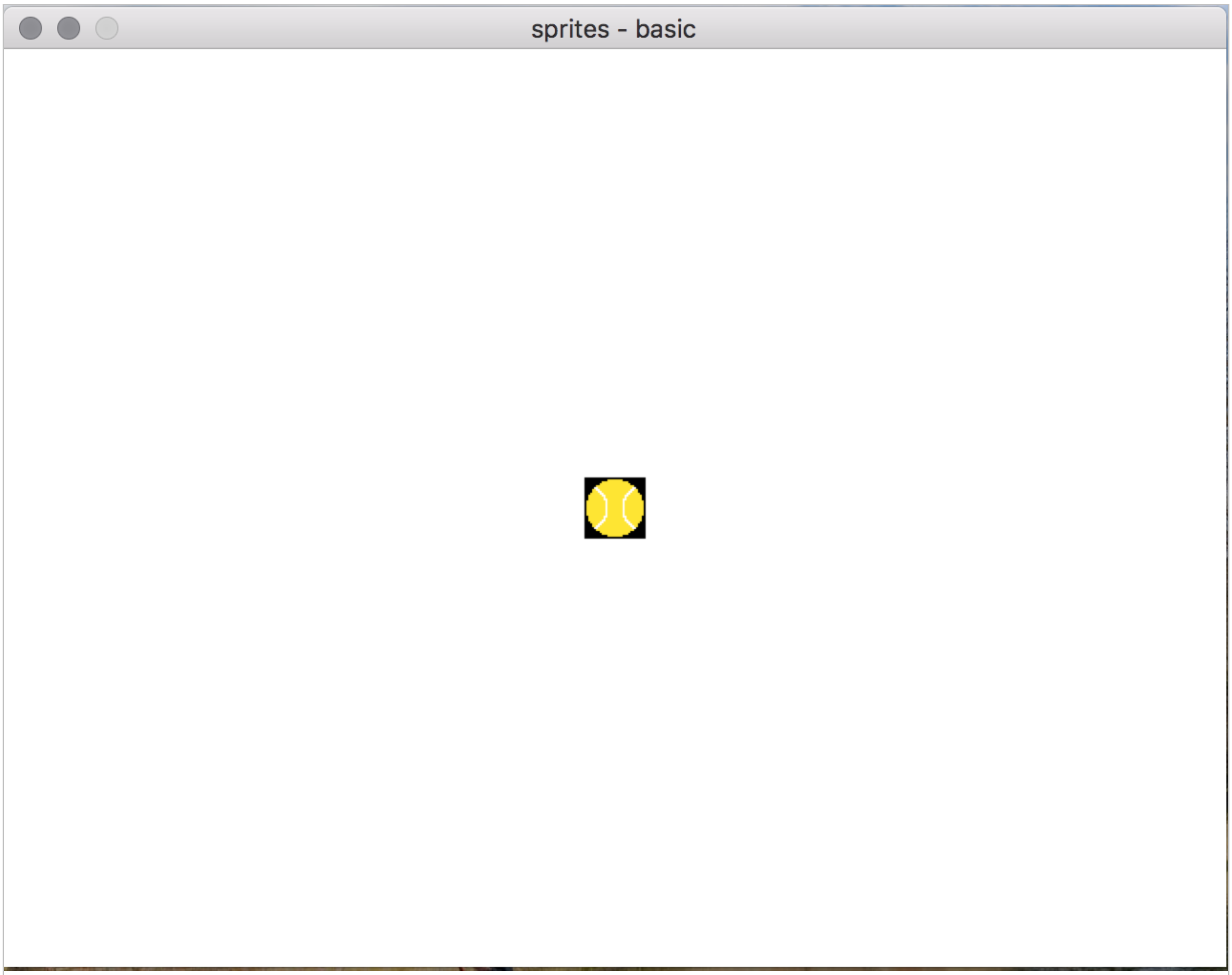


Image sprite

Image - Image Sprite

add transparency

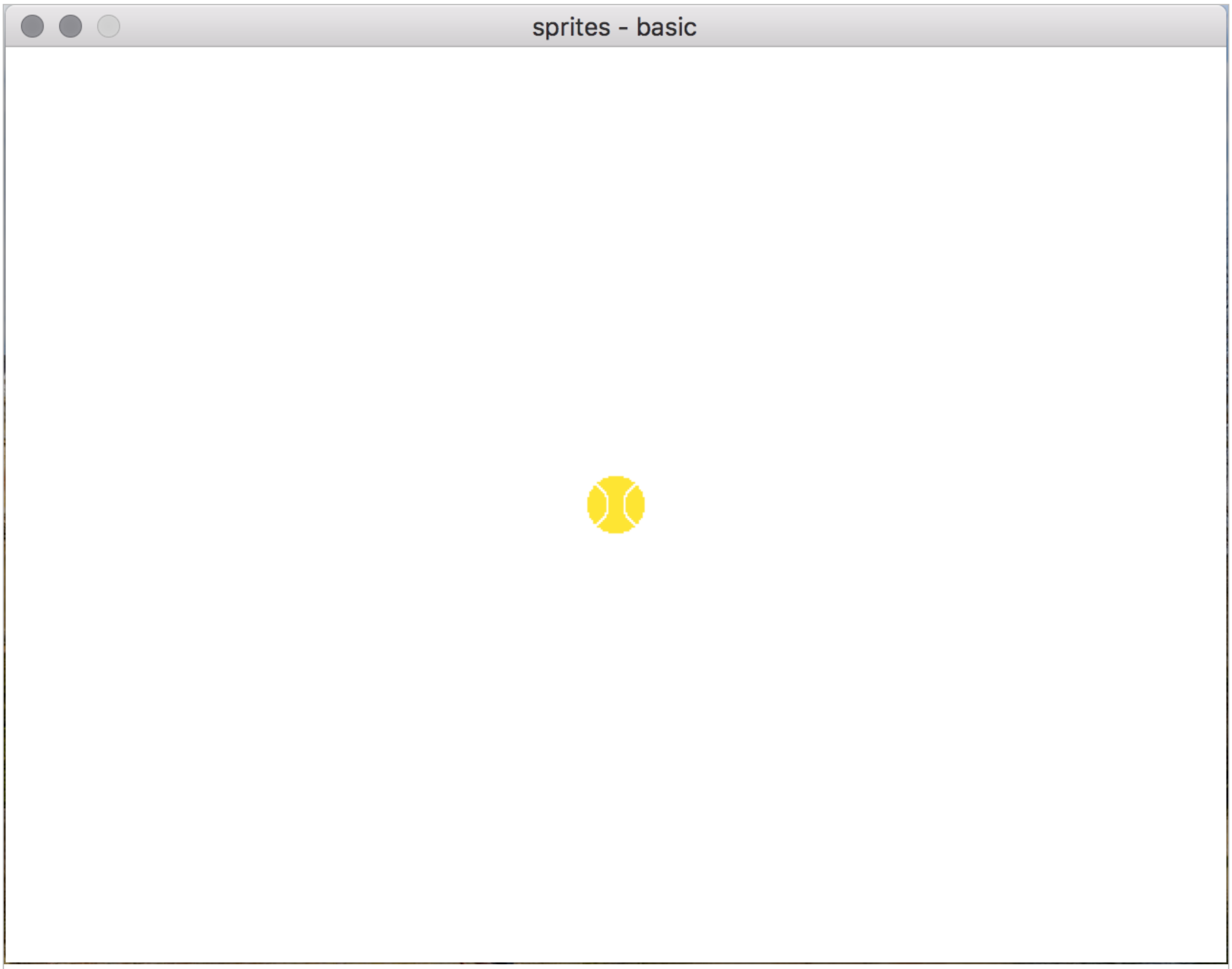
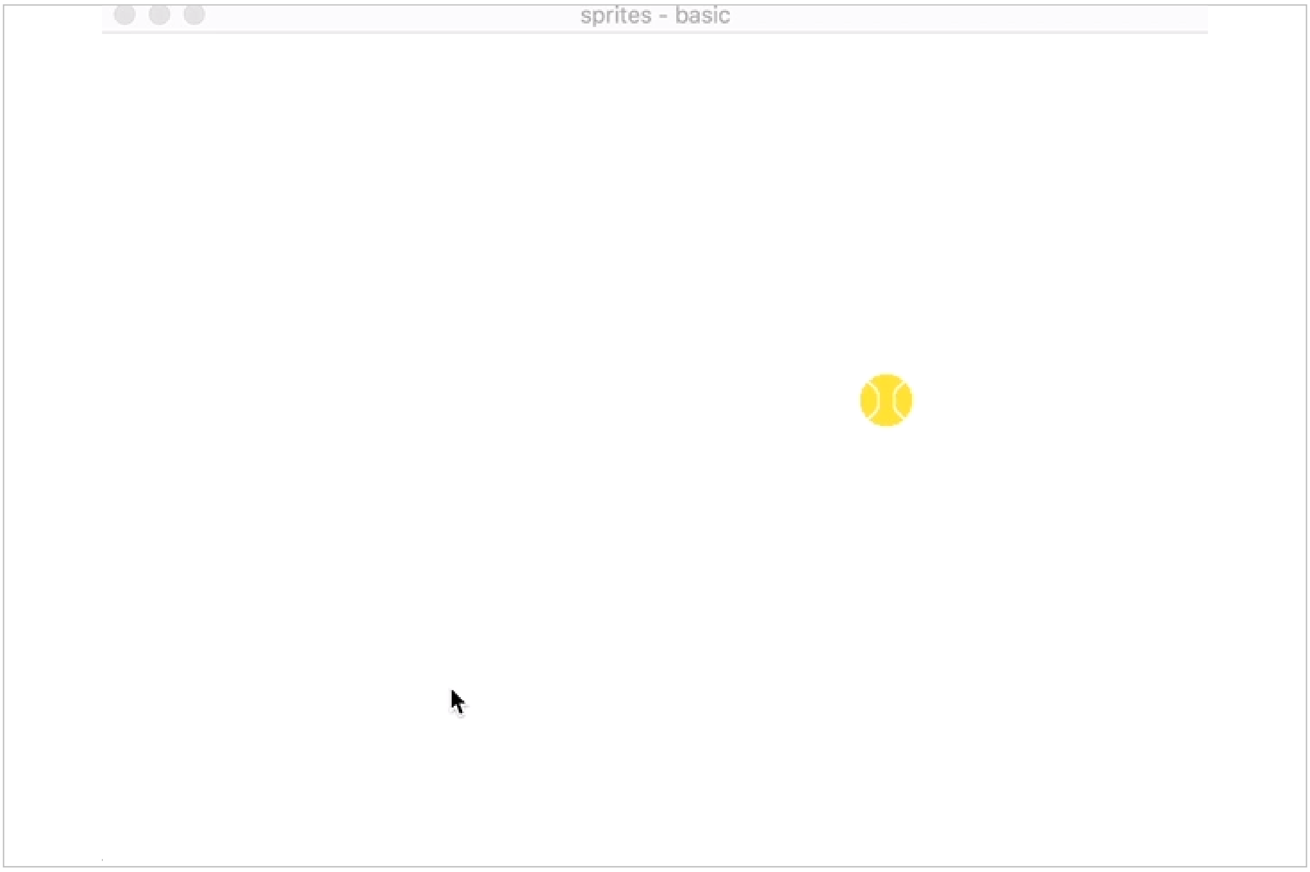


Image sprite - transparent background

Video - Image Sprite

bouncing ball



Python and Pygame - Sprites

control and move, add events...

Please consult the extra notes on Pygame Sprites,

- sprites - control

resources

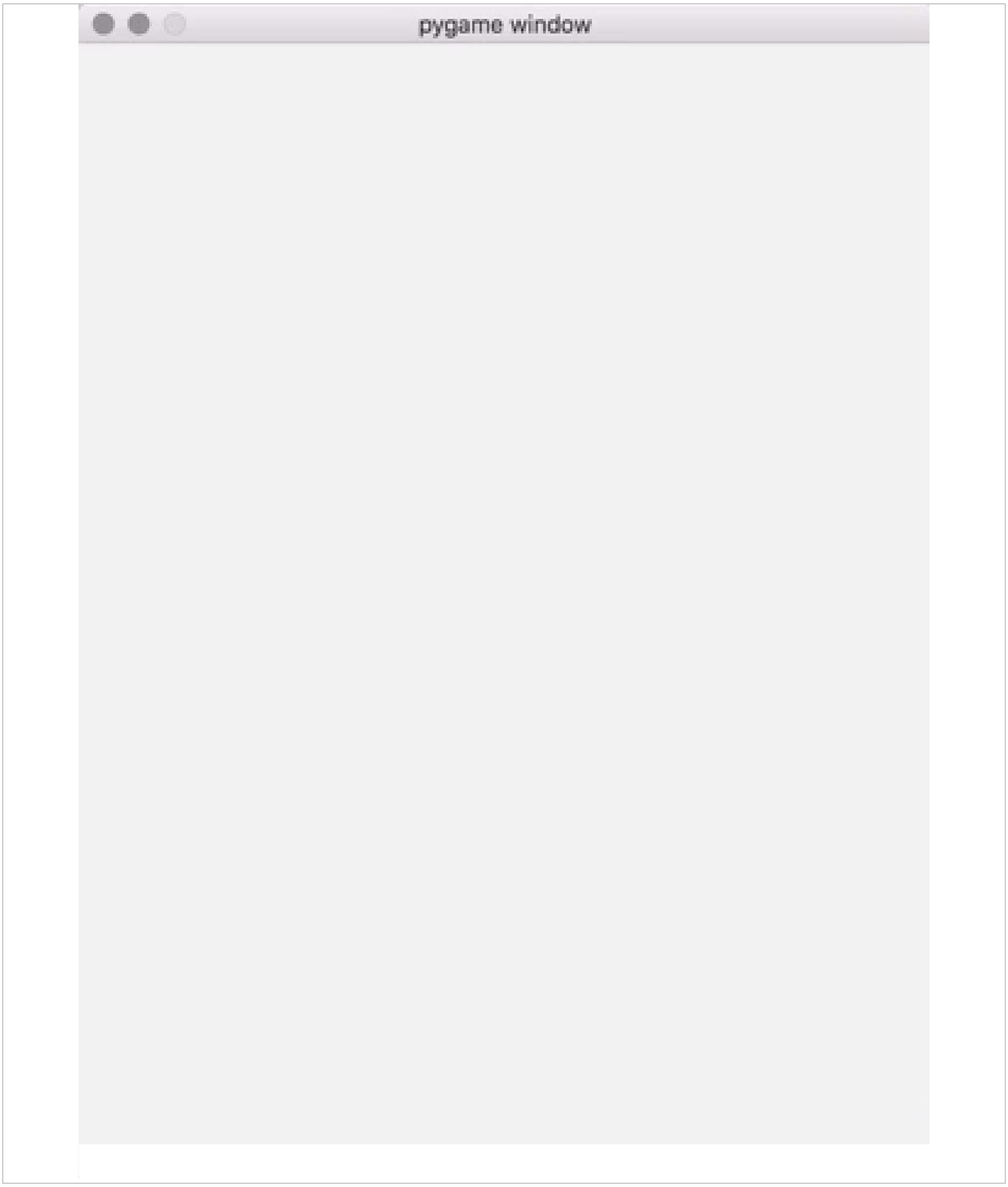
- notes = sprites-control.pdf
- code = basicsprites3.py

game example

- shooter0.1.py - move & control

Video - Shooter 0.1

move & control



Games and formal structure

procedures

- player's consideration and perspective of gameplay and objectives
  - *predicated on a clear understanding of procedures and rules*
- for example,
  - *to be able to act as the player in the chosen game*
  - *to actually know what they can and can't do to complete defined objectives*
- procedures allow us as designers and developers to clearly define
  - *how the player may interact with the game*
  - *and modify the interactive nature of the game*
- e.g. in *Draughts*, each player is allowed to
  - *pick up their own pieces*
  - *then physically move them around the board*
  - *they may also stack pieces*
  - *remove their opponents pieces...*
- e.g. in *Space Invaders*, each player may interact with a physical device
  - *to control their spaceship*
  - *fire their cannon*
  - *select game options...*
- such procedures may be abstracted from the game specific rules

Games and formal structure

rules

- a game's rules may be simple or complex
  - *sometimes to the point of a short novel*
  - *but their intention still remains the same*
- creating a set of clearly defined parameters
  - *what a player can and cannot do to achieve the game's objectives*
- rules may also be used to clarify
  - *what does and does not happen when patterns are matched in a game*
- e.g. in *Draughts*, by completing a certain move
- e.g. in *Space Invaders*, by successfully killing all of the advancing aliens
- some of these rules may be used to define objects
  - *such as the pieces in Draughts or the weapons in Space Invaders*
- others may deal with gameplay concepts
- the very nature of procedures and rules infers a sense of authority
  - *they still require additional structures to enforce them within the game*

Games and formal structure

boundaries

- boundaries help us enforce certain procedures and rules
- using boundaries, to some extent, we may ensure that players of our game
  - *need to adhere to rules to be able complete their objectives*
- e.g. in *Space Invaders*, such boundaries may be physical or digital
  - *restricting the player to a given interaction option*
  - *or certain scope or movement in a game's level*
- such boundaries are creating the imaginary realm of the game
  - *where the rules apply to affect the game's objectives.*
- boundaries help us create the immersive nature of the game
- consider VR and AR
  - *we start to see how new boundaries modify our perceptions*
  - *perceptions of procedures, rules, and gameplay itself*

Games and formal structure

conflict, challenge, battle...

- conflict will often be an active part of playing a game
  - *due to certain objectives within our game*
  - *an indirect consequence of rules we define for the game*
- may also occur in both single player and multi-player games
  - *it will necessarily manifest in different ways*
- we may create such conflict using defined structures of the game
  - *challenging the player with the underlying procedures and rules*
- as a player masters a given part of the game
  - *the conflict will then start to diminish*
  - *or simply be replaced by another problem or situation to resolve*
- e.g. in *Draughts*, initially faced with a direct conflict between players
  - *by simply moving and positioning pieces one player against another*
  - *then, one player starts taking another player's pieces...*
- rules of the game have created the potential for conflict
  - *each player directly challenges the other by leveraging available rules*
- such conflict is another useful tool for modifying gameplay
  - *then modifying difficulty and challenges as a player progresses through a game*
- objectives of a player often conflict with the rules and procedures
  - *may often intentionally limit and guide behaviour within a game*
- by resolving such conflict
  - *a player is able to achieve their desired objectives*
  - *hopefully, the game's overall object as well*

Games and formal structure

outcome, end result...

- another noticeable similarity between games
  - *the simple opportunity for an outcome*
- may include a defined winner, a loser, a draw...
  - *even the simple fixed ending of a story, saga or quest*
- some games may represent such an outcome and end result as either
  - *stay alive and win or die and lose*
- such outcomes may often be a natural conclusion to the defined rules
  - *and the primary, over-arching objective of the game itself*
- however, it doesn't always need to be so clear cut
  - *the end of one adventure, but the beginning of another*
  - *Tolkien-esque in scope and consideration*
- also clear distinction between a game's various objectives and defined outcome
- e.g. in *Space Invaders*, we may see many objectives for a player
  - *destroying aliens, maintaining lives, advancing through different levels...*
- in *Space Invaders*, the single outcome is to
  - *successfully complete each level to complete the game*
- how we use such objectives towards the overall outcome
  - *is an option we can use to modify gameplay itself*
  - *and the overall experience of our game*
- in multi-player games, a key component of a game's outcome
  - *includes the palpable sense of uncertainty*
- as we increase conflict and competition
  - *uncertainty will likewise be increased*
  - *becomes a key factor in encouraging player's to return to a game*

Image - Create a memorable ending

Super Mario Bros. vs Castlevania

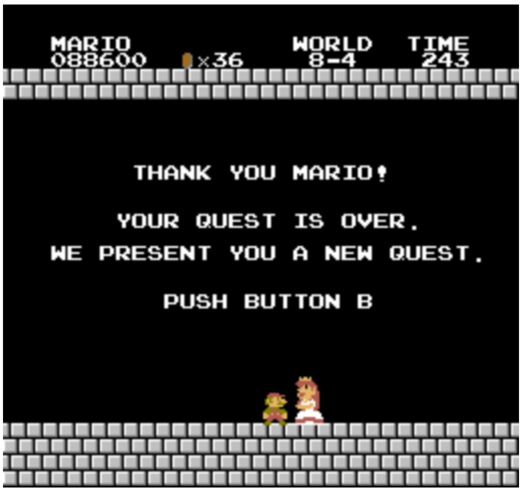
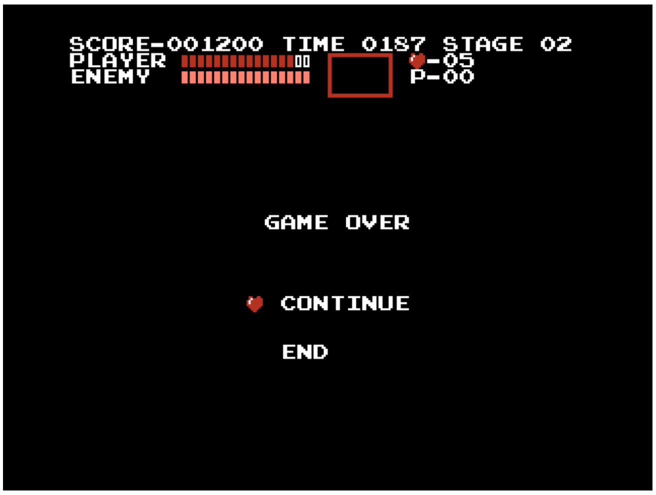
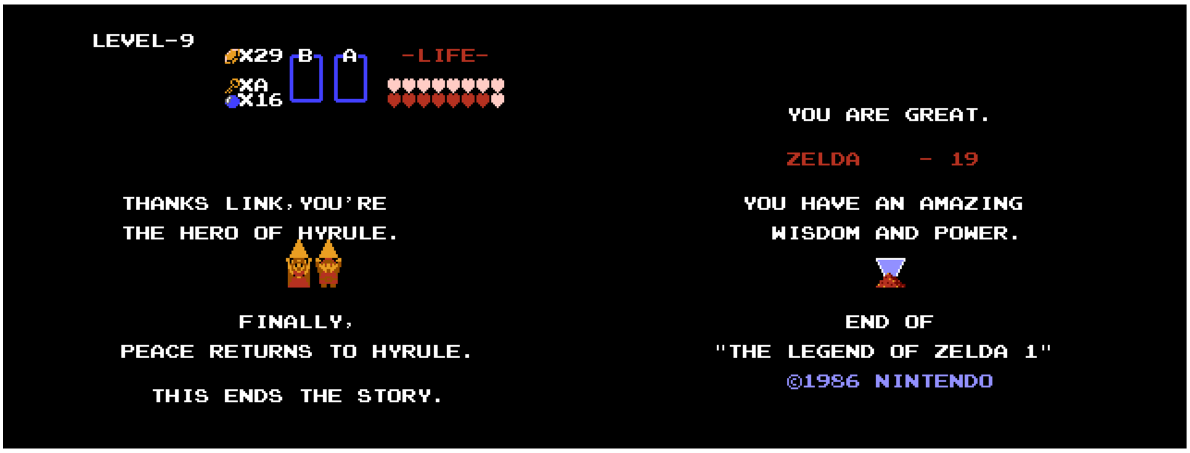
Super Mario Bros.	Castlevania
 MARIO WORLD TIME 088600 ×36 8-4 243 THANK YOU MARIO! YOUR QUEST IS OVER. WE PRESENT YOU A NEW QUEST. PUSH BUTTON B	 SCORE-001200 TIME 0187 STAGE 02 PLAYER ♥-05 ENEMY P-00 GAME OVER ♥ CONTINUE END

Image - Create a memorable ending

Legend of Zelda

Legend of Zelda



THANKS LINK, YOU'RE
THE HERO OF HYRULE.

FINALLY,
PEACE RETURNS TO HYRULE.

THIS ENDS THE STORY.

YOU ARE GREAT.

ZELDA - 19

YOU HAVE AN AMAZING
WISDOM AND POWER.

END OF
"THE LEGEND OF ZELDA 1"
©1986 NINTENDO

Game example - Space Invaders

a classic bit of fun...

- Space Invaders - Sega and Taito
  - *close fidelity example from 1985 - graphics almost identical to original 1979 version released in Japan*
  - *streaming version of game*
- Draughts/Checkers
  - *playable version*

Python and Pygame - Game Example I

shooter style game - STG

- start creating our first full game example
  - *shooter example - **STG** in Japan*
- this game will help us design, develop, and test the following:
  - *user control*
  - *enemy objects*
  - *collision detection*
  - *firing projectiles at enemies*
  - *destroying enemy objects*
  - *add custom sprites and graphics*
  - *improve the collision detection*
  - *start animating sprite images*
  - *radomise enemy objects to create greater challenges*
  - *keep a running game score and render to game window*
  - *add game music and sound effects*
  - *check our player's health...*
  - *add some fun game extras*
    - e.g. health status, explosions...
    - lots more...

Python and Pygame - Game Example I

add more objects - mob

- now start to add extra sprite objects to our game window
  - *commonly given a collective, generic name of* ***mob***
- add the following class `Mob` to our game

```
# create a generic mob sprite for the game - standard name is *mob*
class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((20, 20))
        self.image.fill(CYAN)
        # specify bounding rect for sprite
        self.rect = self.image.get_rect()
        # specify random start posn & speed of enemies
        self.rect.x = random.randrange(winWidth - self.rect.width)
        self.rect.y = random.randrange(-100, -50)
        self.speed_y = random.randrange(1, 10)

    def update(self):
        self.rect.y += self.speed_y
```

- with this class we can create extra sprite objects
  - *set their size, colour, &c.*
  - *then set random `x` and `y` coordinates for the starting position of the sprite object*
- use random values to ensure that the objects start and move from different positions
  - *from the top of the game window*
  - *then progress in staggered groups down the window...*

Python and Pygame - Game Example 1

update extra objects

- as our enemy objects move down the game window
  - *need to check if and when they leave the bottom of the game window*
- we can add the following checks to the `update` function

```python
def update(self):
    self.rect.y += self.speed_y
    # check if enemy sprite leaves the bottom of the game window - then randomise at the top...
    if self.rect.top > winHeight + 15:
        # specify random start posn & speed of enemies
        self.rect.x = random.randrange(winWidth - self.rect.width)
        self.rect.y = random.randrange(-100, -50)
        self.speed_y = random.randrange(1, 7)
```

- as each sprite object leaves the bottom of the game window
  - *we can check its position*
- then, we may reset the sprite object to the top of the game window
- need to ensure that the same sprite object does not simply loop around
  - *and then reappear at the same position at the top of the game window*
  - *becomes too easy and tedious for our player...*
- instead, we can reset our *mob* object to a random path down the window
  - *should make it slightly harder for our player*
- also ensure that each extra sprite object has a different speed
  - *by simply randomising the speed along the y-axis per sprite object*

Python and Pygame - Game Example 1

show extra objects

- now create a *mob* group as a container for our extra sprite objects
- group will become particularly useful as we add collision detection later in the game
  - *update our code as follows, e.g.*

```
# sprite groups - game, mob...
mob_sprites = pygame.sprite.Group()
# create sprite objects, add to sprite groups...
for i in range(10):
    mob = Mob()
    # add to game_sprites group to get object updated
    game_sprites.add(mob)
    # add to mob_sprites group - use for collision detection &c.
    mob_sprites.add(mob)
```

- create our *mob* objects
  - *then add them to the required sprite groups*
- by adding them to the `game_sprites` group
  - *they will be updated as the game loop is executed*
- `mob_sprites` group will help us easily detect extra sprite objects
  - *e.g. when we need to add collision detection*
  - *or remove them from the game window...*

Python and Pygame - Game Example I

modify motion of extra objects

- above updates work great for random motion along the y-axis
  - *add some variation to movement of extra sprite object by modifying the x-axis*
- we can modify the x-axis for each extra sprite object
  - *creates variant angular motion along both the x-axis and y-axis, e.g.*

```
# random speed along the x-axis
self.speed_x = random.randrange(-3, 3)
...

self.rect.x += self.speed_x
# check if sprite leaves the bottom of the game window - then randomise at the top...
if self.rect.top > winHeight + 15 or self.rect.left < -15 or self.rect.right > winWidth + 15:
    # specify random start posn & speed of extra sprite objects
    self.rect.x = random.randrange(winWidth - self.rect.width)
    self.speed_x = random.randrange(-3, 3)
...
```

Python and Pygame - Game Example I

modify motion of extra objects - continued

- our mob class may now be updated as follows,

```
# create a generic extra sprite object for the game - standard name is *mob*
class Mob(pygame.sprite.Sprite):
  def __init__(self):
      pygame.sprite.Sprite.__init__(self)
      self.image = pygame.Surface((20, 20))
      self.image.fill(CYAN)
      # specify bounding rect for sprite
      self.rect = self.image.get_rect()
      # specify random start posn & speed
      self.rect.x = random.randrange(winWidth - self.rect.width)
      self.rect.y = random.randrange(-100, -50)
      # random speed along the x-axis
      self.speed_x = random.randrange(-3, 3)
      # random speed along the y-axis
      self.speed_y = random.randrange(1, 7)

  def update(self):
      self.rect.x += self.speed_x
      self.rect.y += self.speed_y
      # check if sprite leaves the bottom of the game window - then randomise at the top...
      if self.rect.top > winHeight + 15 or self.rect.left < -15 or self.rect.right > winWidth + 15:
          # specify random start posn & speed of extra sprite objects
          self.rect.x = random.randrange(winWidth - self.rect.width)
          self.rect.y = random.randrange(-100, -50)
          self.speed_x = random.randrange(-3, 3)
          self.speed_y = random.randrange(1, 7)
```

- added a quick check for motion of our extra sprite object along the x-axis
  - *as sprite exits on either side of the screen*
  - *create a new sprite on a random path down the screen*

resources

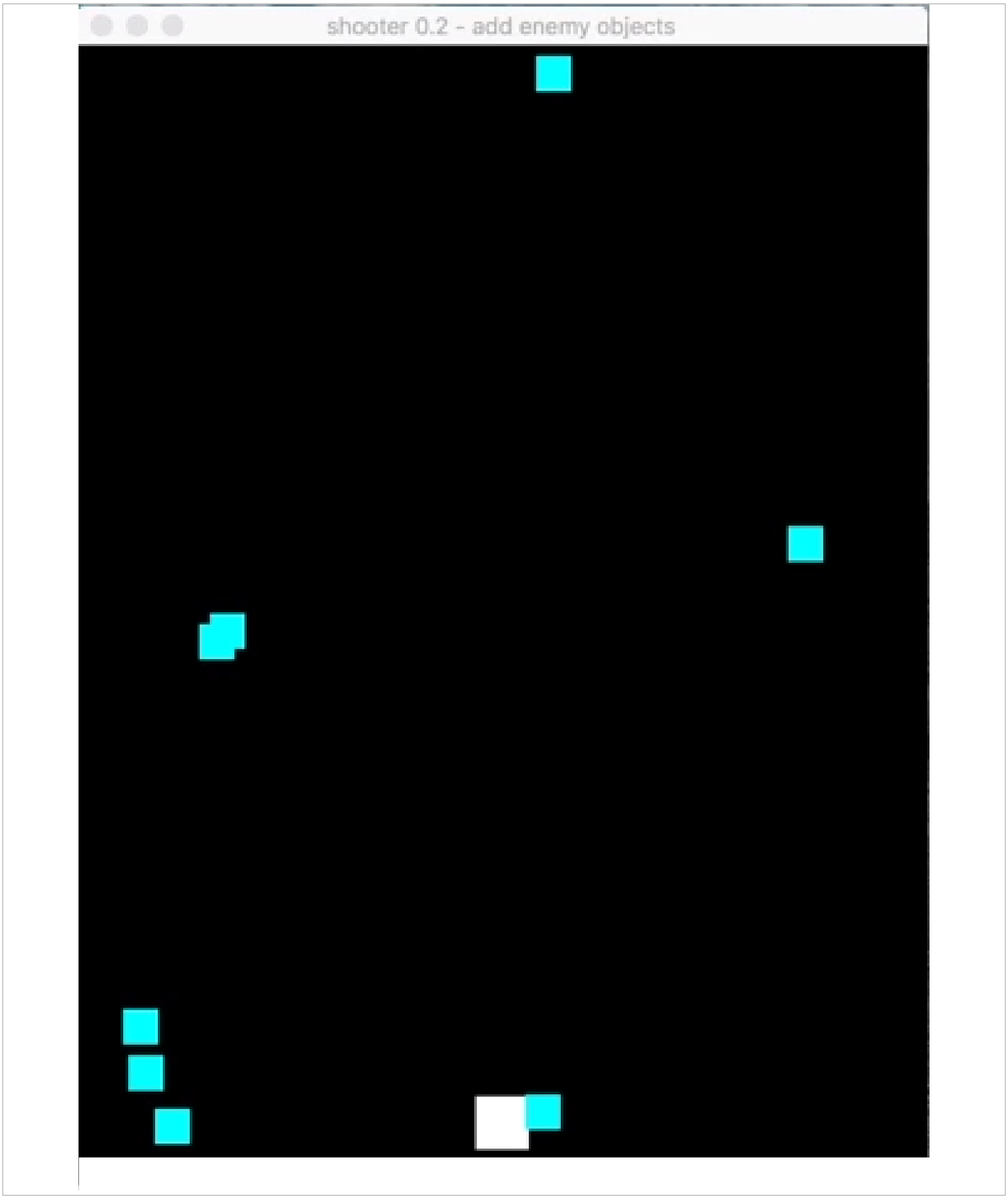
- notes = sprites-more-objects.pdf
- code = basicsprites4.py

game example

- shooter0.2.py
  - *add enemy objects*

Video - Shooter 0.2

move & control



Python and Pygame - Game Example I

add new sprites

- create a new class for this sprite object
  - *e.g. projectiles that a player may appear to fire from the top of player object*
  - *such as a ship &c*

```python
# create a generic projectile sprite - for bullets, lasers &c.
class Projectile(pygame.sprite.Sprite):
    # x, y - add specific location for object relative to player sprite
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((5, 10))
        self.image.fill(RED)
        self.rect = self.image.get_rect()
        # weapon fired from front (top) of player sprite...
        self.rect.bottom = y
        self.rect.centerx = x
        # speed of projectile up the screen
        self.speed_y = -10

    def update(self):
        # update y relative to speed of projectile on y-axis
        self.rect.y += self.speed_y
        # remove from game window - if it goes beyond bounding for y-axis at top...
        if self.rect.bottom < 0:
            # kill() removes specified sprite from group...
            self.kill()
```

- creating another sprite object for a projectile such as a bullet or a laser beam
- projectile will be shot from the top of another object
  - *set `x` and `y` coordinates relative to position of player's object*
  - *setting the speed along the y-axis so it travels up the screen*
- as we update each projectile object
  - *update its speed, and then check its position on the screen...*
- if it leaves the top of the game window
  - *we can call the generic `kill()` method on this sprite*
- method is available for any sprite object we create in the game window

Python and Pygame - Game Example 1

listen for keypress

- need to add a new listener to the game loop to detect a keypress for the *spacebar*
- use this keypress to allow a player to shoot these projectiles, e.g. a laser beam

```python
# 'processing' inputs (events)
for event in EVENTS.get():
    # check keyboard events - keydown
    if event.type == pygame.KEYDOWN:
        # check for ESCAPE key
        if event.key == pygame.K_ESCAPE:
            gameExit()
        elif event.key == pygame.K_SPACE:
            # fire laser beam...
            player.fire()
```

- updated our keypress listeners to check each time a player hits down on the *spacebar*
- use this keypress event to fire our projectile
  - *e.g. a laser beam to hit our enemy mobs...*

Python and Pygame - Game Example I

release new sprites

- as player hits the *spacebar*, we need to create new sprites
- new sprite objects will then be released from the top of the player's object
- relative position of one sprite object is determining start position of another sprite object
- need to update the class for our primary sprite object, e.g. a player
  - *include a method for firing the projectiles from the top of this sprite object, e.g.*

```
# fire projectile from top of player sprite object
def fire(self):
    # set position of projectile relative to player's object rect for centerx and top
    projectile = Projectile(self.rect.centerx, self.rect.top)
    # add projectile to game sprites group
    game_sprites.add(projectile)
    # add each projectile to sprite group for all projectiles
    projectiles.add(projectile)
```

- sets start position for x and y coordinates of each projectile sprite
  - *sets to the current position of the player's sprite object*
- then, add each projectile sprite object to the main game sprite group
  - *and add a new sprite group for all of the projectiles*
  - *add this new sprite group as follows,*

```
projectiles = pygame.sprite.Group()
```

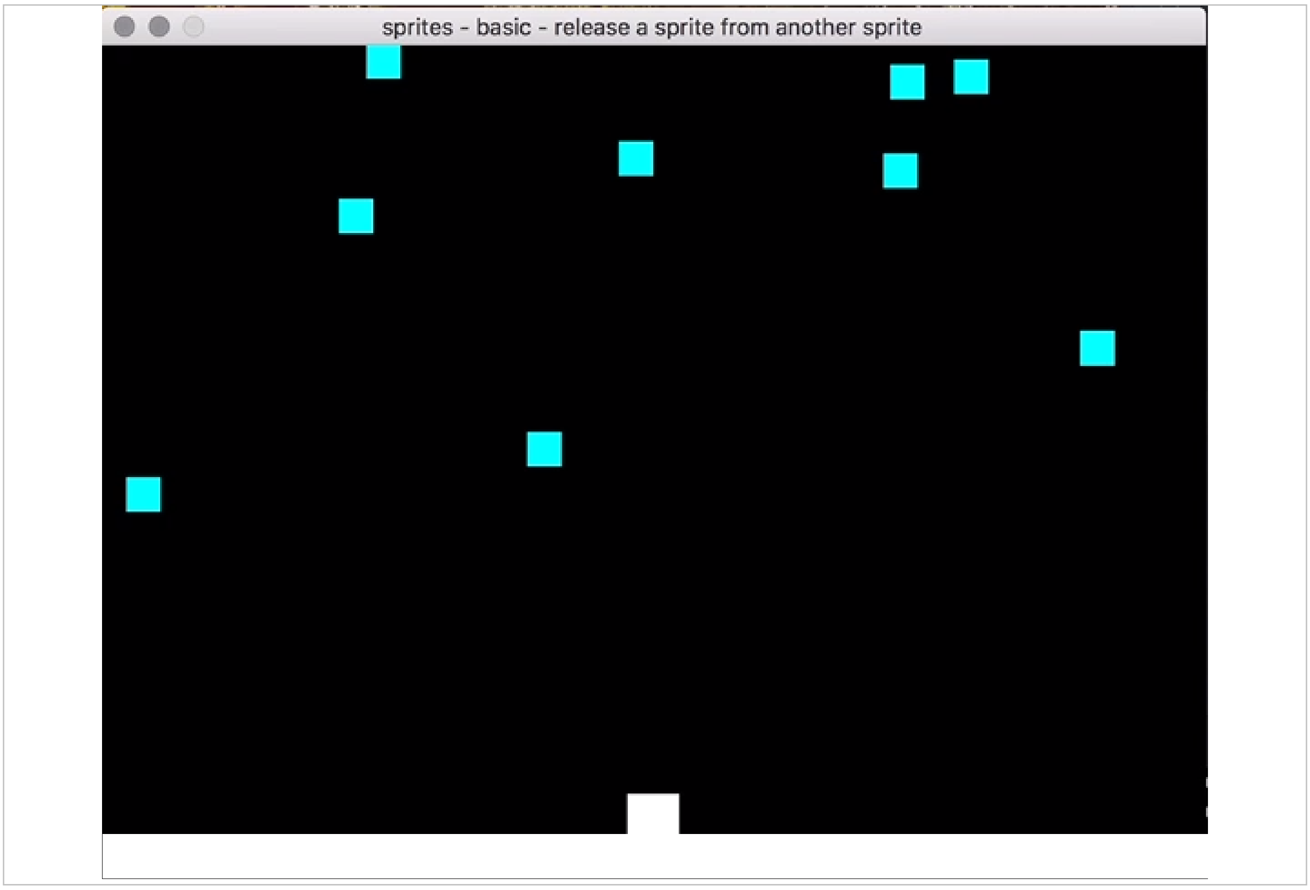
- when a player presses down on the *spacebar* a projectile will be fired
  - *a red laser beam from the top of the player's sprite object*

resources

- notes = sprites-relative-objects.pdf
- code = basicsprites5.py

Video - Basic Sprites

relative objects



Python and Pygame - Game Example 1

basic collision detection

- Pygame includes support for adding explicit collision detection
  - *between two or more sprites in a game window*
  - *use built-in functions to help us work with these collisions*
- add basic collision detection
  - *each time an object hits the player's object at the foot of the game window*
  - *Pygame includes the following function, e.g.*

```
# add check for collision - extra objects and player sprites (False = hit object is not deleted from game wi
pygame.sprite.spritecollide(player, mob_sprites, False)
```

- sprite object's function allows us to check if one sprite object has been hit by another
- e.g. checking if `player` sprite object hit by another sprite object
  - *in this example, from the `mob_sprites` group*
- `False` parameter is a boolean value for the state of the object that has hit
  - *i.e. determines whether a mob sprite object should be deleted from game window or not*
- particularly useful as it returns a *list* data structure
  - *contains any mob sprite objects that hit the player sprite object*
  - *update this code as follows, and store this list in a variable, e.g.*

```
collisions = pygame.sprite.spritecollide(player, mob_sprites, False)
```

- then use this *list* to check if any collisions have occurred in our game window, e.g.

```
...
if collisions:
  # update game objects &c.
  ...
...
```

- use boolean value to check if the *list* `collisions` is empty or not

Python and Pygame - Game Example I

Sprite group collision detection

- now add collision detection for various groups of sprites
  - *e.g. one group of sprites may be colliding with another, defined sprite group...*
- use Pygame's collide method for sprite groups, e.g.

```
# add check for sprite group collide with another sprite group - projectiles hitting enemy objects - use Tru
collisions = pygame.sprite.groupcollide(mob_sprites, projectiles, True, True)
```

- boolean parameter values of `True` and `True`
  - *allow us to delete both the hit enemy objects*
  - *and the projectile objects that hit them*
- as *list* of `collisions` is populated
- create new sprite objects for those that have been hit and deleted
- e.g. extra objects that move down the game window

```
# add more mobs for those hit and deleted by projectiles
for collision in collisions:
    mob = Mob()
    game_sprites.add(mob)
    mob_sprites.add(mob)
```

- if we don't create new extra objects
  - *game window will quickly run out of sprite objects*

resources

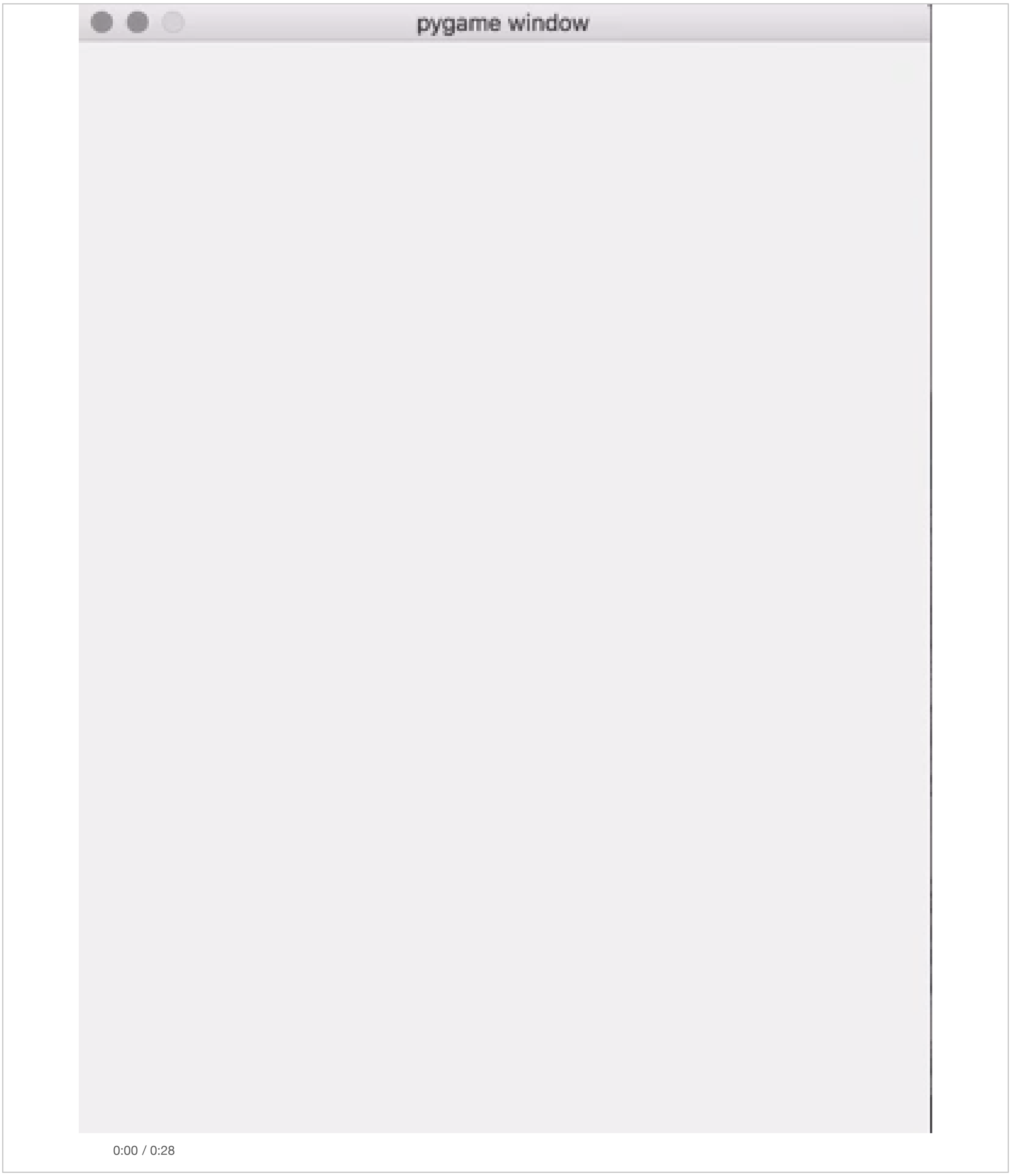
- notes = sprites-collision-detection.pdf
- code = basicsprites6.py

game example

- shooter0.3.py
  - *collision detection of single sprite*
  - *detect group collisions*

Video - Shooter0.3

basic collisions and firing



Resources

Demos

- pygame sprites - basic
  - *basicsprites1.py*
  - *basicsprites2.py*
  - *basicsprites3.py*
  - *basicsprites4.py*
  - *basicsprites5.py*
  - *basicsprites6.py*
- pygame collision detection - basic
  - *collisionsprites1.py*
  - *collisionsprites2.py*
- pygame - Game 1 Example
  - *shooter0.1.py*
  - *shooter0.2.py*
  - *shooter0.3.py*

Games

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

Game notes

- Pygame
  - *sprites-intro.pdf*
  - *sprites-set-image.pdf*
  - *sprites-control.pdf*
  - *sprites-more-objects.pdf*
  - *sprites-relative-objects.pdf*
  - *sprites-collision-detection.pdf*

References

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