Comp 388/488 - Introduction to Game Design and Development

Spring Semester 2017 - Week 11

Dr Nick Hayward

Contents

Game Dev resources

music, sound effects...

- for a game's sound effects
  - *many different options and sources for these sounds*
- try open source examples, e.g.
  - *Open Game Art*
- perhaps create our own custom sounds using a utility such as **SFXR**, e.g.
  - *SFXR*
- or its derivative website option, e.g.
  - *BFXR*

Games and formal elements

intro

- as with each formal structure
  - *players*
  - *objectives*
  - *procedures & rules*
    - including implied **boundaries**
  - *conflict, challenge, battle...*
  - *outcome, end result...*
- these constituent elements come together
  - *to form what we largely understand to be a game*
- such formal elements constitute how we
  - *design*
  - *structure*
  - *develop our video games*
- overlap and interplay of these formal elements
  - *has now become the foundation for game design*
- a sound understanding and knowledge of these formal elements
  - *their usage and application*
  - *helps us start creating innovative, playful game experiences*

Games and formal elements

players and games - part 1

- identified the need for rules, procedures...
- within the confines of such rules
  - *players are suspending normal societal restrictions*
  - *players enact shooting, fighting, role-play, and magical roles...*
  - *roles, actions &c. normally confined to a passive medium, e.g. books, film...*
- such actions can often form stark contrasts in a game environment
- rules become enacted in a **magic circle**
  - *originally described by Huizinga in his 1938 title,* ***Homo Ludens***
  - *later adapted, and refashioned for digital games*

> *In a very basic sense, the magic circle of a game is where the game takes place. To play a game means entering into a magic circle, or perhaps creating one as a game begins.*

Salen, K. & Zimmerman, E. *Rules of Play: Game Design Fundamentals*. MIT Press. 2003.

Games and formal elements

players and games - part 2

- such rules naturally create the opportunity for play
  - *within the defined confines of a game*
- our use of rules, characters, story, and even mechanics and control
  - *invites a player to become involved with, and invested in, our game*
- motion controllers became an invitation for players to intuitively enter a game
  - *e.g. Nintendo's Wii, Microsoft's Xbox Kinect, and Sony's Playstation Move...*
- the premise of many games now became an extension of the controller
- it's not only a matter of engaging and inviting players into your game
- need to consider the nature and structure of player participation, e.g.
  - *how many players does the game support?*
  - *will each player adopt the same role?*
  - *perhaps play in a team or in direct competition*
- how we answer such questions will have a direct influence
  - *on the nature of the game we're designing*
  - *its gameplay*
  - *and a player's engagement with the story and characters...*

Image - Motion Controllers

Nintendo Wii	Xbox Kinect	Playstation Move
		

Video - Xbox Kinect & Algorithms

Kinect algorithm starts at 47:36 minutes into the video.

Source - Algorithms, YouTube

Games and formal elements

players, patterns, and numbers

- games may be designed and developed for a variety of player numbers
  - *from strict single player options to varied multiplayer environments*
  - *MMOG push player numbers to the upper bounds*
- player numbers will often determine patterns of interaction for your game
- patterns may include
  - *single player versus the game*
    - e.g. Space Invaders, Mario, Sonic...
  - *multiple individual players versus the game*
    - e.g. sports, racing, card games...
  - *single player versus another player*
    - e.g. games such as Street Fighter and Mortal Kombat...
  - *multiple players versus a single player*
    - many detective and role playing board games include such features
    - some *god* games may also be structured using this pattern
  - *collaborative play*
    - players work together against the game
    - sports games, *Journey* by designer Jenova Chen...
  - *multiple players competing*
    - e.g. Halo, Call of Duty...
    - standard pattern referenced for *multiplayer* games
  - *team competitions...*
    - e.g. eSports such as League of Legends...

Games and Dynamics

systems and evolution - intro

- a game's players, their type, number, reactions, behaviours &c.
  - *may also be a reflection of the game system itself*
- systems may often display complex and unpredictable results
  - *e.g. when set in motion as part of the broader, general gameplay*
- such systems are not inherently predicated on complexity and scope
- good examples of simple rule sets and patterns
  - *produce unpredictable results as they are set in motion*
- we may see such patterns on a regular basis, e.g. in the natural world
  - *complex systems emerging due to collaboratrive structures, e.g.*
  - *ant colony*
  - *bee hive and pollen collection*
  - *swarm intelligence of a confusion of wildebeest*
- human consciousness may be the product of such systems
  - *commonly referred to as emergent systems*
- well-known experiment in emergence was conducted by John Conway in the 1960s
  - *particularly useful and interesting for us as game designers and developers*
- Conway was particularly curious about the working systems of rudimentary elements
  - *how did such elements work together based upon a set of defined, simple rules...*
- he wanted to clearly demonstrate this phenomenon at its simplest level
  - *e.g. in a defined 2D space, such as a known chess board*
- he tested various ideas and concepts
  - *considered ideas such as on and off logic for squares/cells on a board*
  - *logic based on rules for adjacent squares/cells*
- he continued his experiments and tests
  - *in a similar manner to a game designer*

- *toyed with various sets of rules for several years*

Games and Dynamics

systems and evolution - simple rules

rules

- Birth
  - *if a cell is unpopulated, and surrounded by exactly 3 populated cells, this cell will be populated in the next generation*
- Death by loneliness
  - *if a cell is populated, and surrounded by fewer than 2 populated cells, this cell will be unpopulated in the next generation*
- Death by overpopulation
  - *if a cell is populated, and surrounded by at least 4 populated cells, this cell will be unpopulated in the next generation*
- emergent system finally converged on the above set of rules
- Conway, and some of his colleagues at Cambridge, began populating their chess board with pieces
  - *then tested their rules by hand*
- started to learn about this system
  - *and the very nature of emergent, almost evolving systems*
- quickly realised that different starting conditions had a noticeable impact on a system's evolution
- realised that the complexity of such start conditions might have a side-effect on the patterns created
  - *many simply failing to survive and evolve*
- a particularly interesting discovery became known as the **R Pentomino** configuration
  - *Richard Guy, an associate of Conway, became fascinated with this particular configuration*
- Guy tested their defined rules for more than a hundred generations
  - *started to observe various patterns emerging*
- a regular mix of shapes and patterns emerging

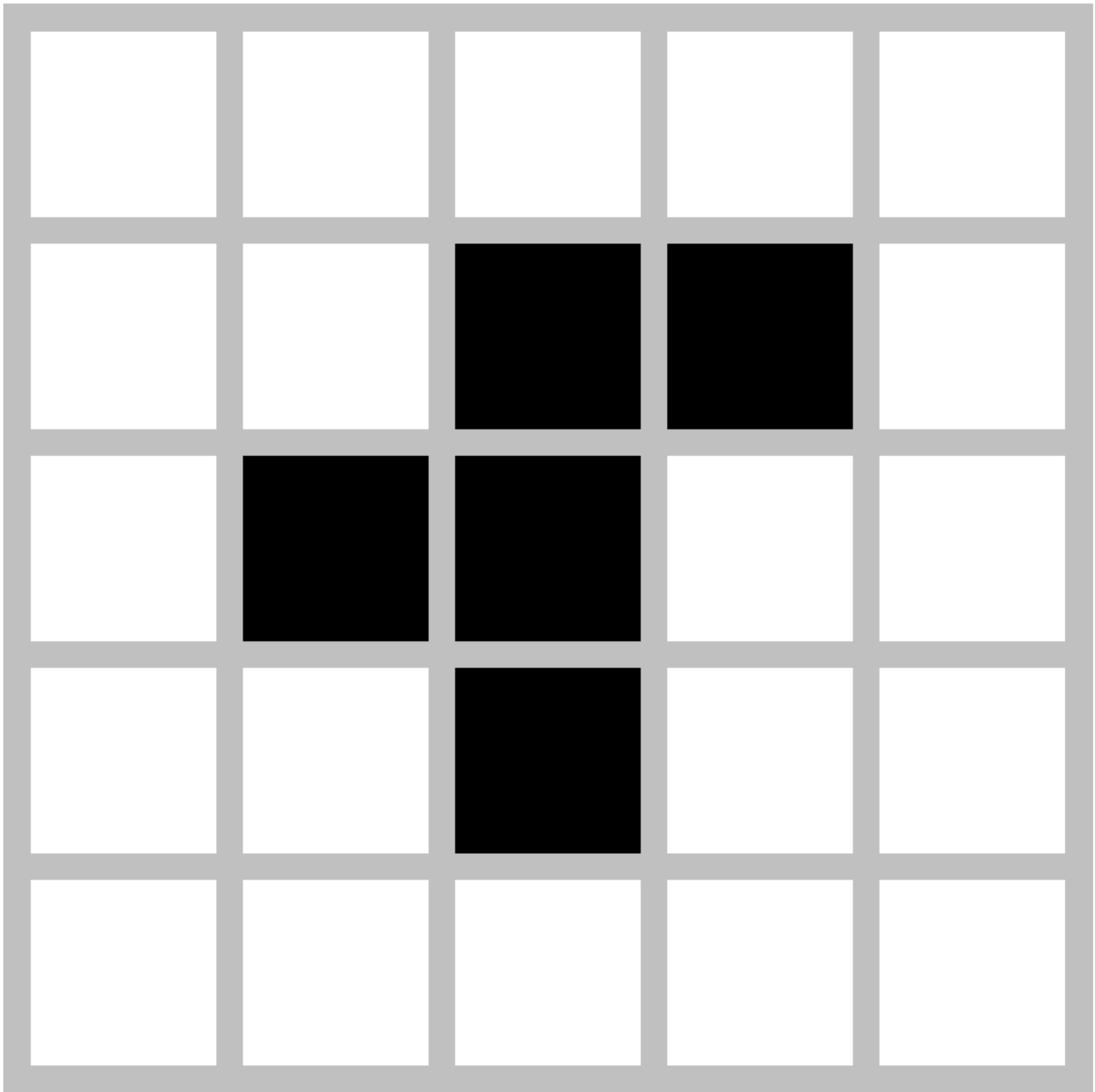
- *Guy noticed that his shapes appeared to moving, effectively walking across the board*
- *he exclaimed at this discovery,*

Look, my bit's walking

- Poundstone, W. *Prisoner's Dilemma.* Touchstone. New York. 2002.
- Guy continued to test and work on this configuration
  - *until he was able to get this pattern to actually walk across the room*
  - *and then out the door...*
- Guy's discovery became known simply as a **glider**

Image - Systems and Evolution

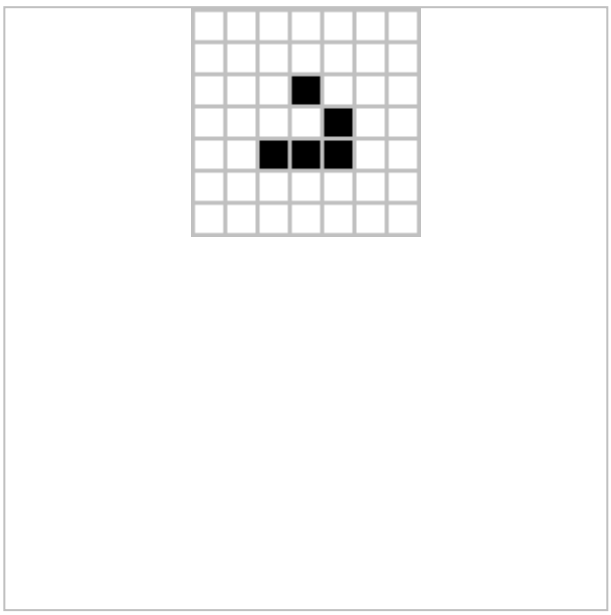
R-Pentomino



R Pentomino

Image - Systems and Evolution

glider evolution



- interactive demo - glider

Games and Dynamics

systems and evolution - examples

- such simple emergent systems demonstrated
  - *benefits and application of simplicity in rules and patterns*
  - *their ability to evolve into life-form style patterns*
- such systems had the potential to evolve and develop with each generation
  - *particularly interesting and useful to us as game designers*
- may start to add such techniques to help make our games
  - *more realistic, evolving, and unpredictable for our players...*
- example games using these techniques include:
  - *Black and White*
  - *Grand Theft Auto (v3 onwards)*
  - *Halo*
  - *Oddworld: Munch's Oddysee*
  - *The Sims*
  - ...

Python and Pygame - Game Example 1

random mob sprite images

- as we add sprite image objects to a game window, e.g. multiple *mob* images
  - *we can make the game experience more fun*
  - *by randomising the image for each mob sprite object*
- we may use a group of images as possible mob images
  - *then randomise their selection for each new mob sprite object image*
- to add random image, at least randomised from potential options...
  - *need to add a list of available images for the random selection, e.g.*

```
asteroid_list = ["asteroid-tiny-grey.png", "asteroid-small-grey.png", "asteroid-med-grey.png"]
```

- also need a new list for our asteroid images, e.g.

```
asteroid_imgs = []
```

- simply need to loop through this asteroid list
  - *then add each available image to the list of* `asteroid_imgs`*, e.g.*

```
for img in asteroid_list:
    asteroid_imgs.append(pygame.image.load(os.path.join(img_dir, img)).convert())
```

- then update the `Mob` class to set a random image from `asteroid_imgs` list, e.g.

```
self.image_original = random.choice(asteroid_imgs)
```

- images for our mob sprite objects will now be randomly chosen from the available list of images

resources

- notes = sprites-animating-random-images.pdf
- code = animatingsprites2.py

game example

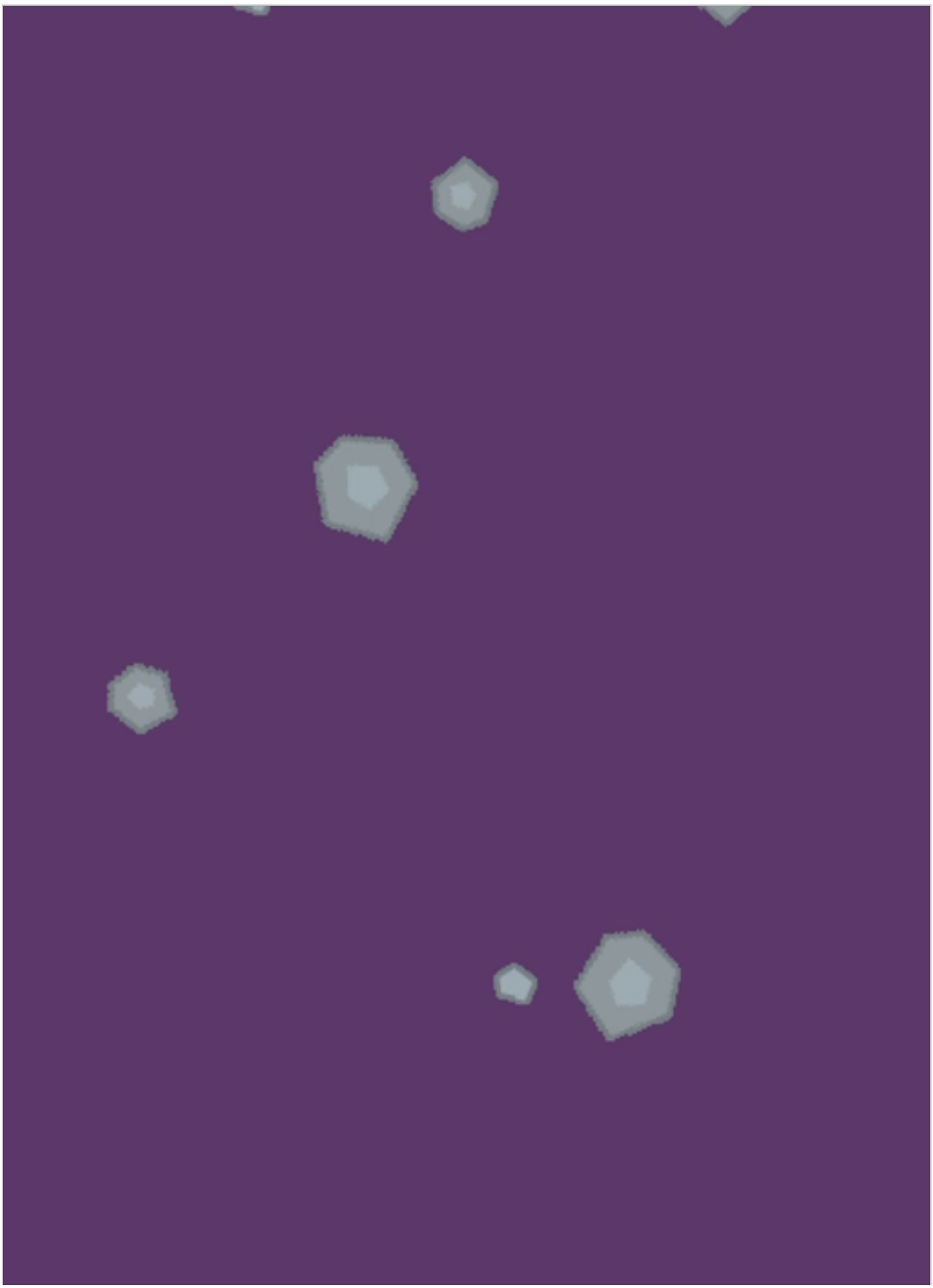
- shooter0.7.py
- set random image for mob sprite object image
  - *random image from selection of image options*
  - *rotate and animate each random mob sprite image*

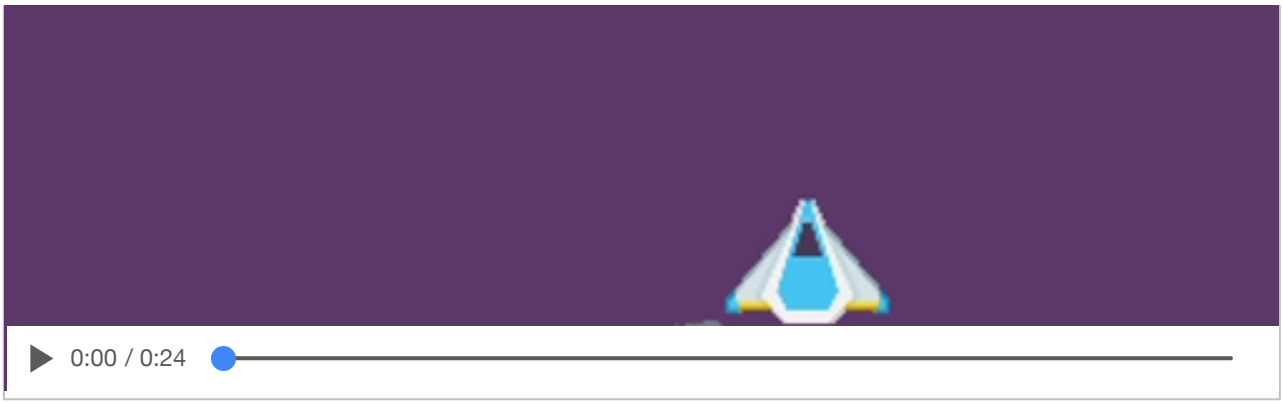
Video - Animating Sprites

random mob images

Video - Shooter 0.7

set random image for mob sprite object image




0:00 / 0:24

Games and development

quick exercise

A quick exercise to consider evolution in systems,

- *Traveling Salesman Problem*
- evolution of simple systems
- swarm/hive intelligence
- interaction in systems

questions

- consider the above, and how they might interact in a system to evolve an optimal solution to a problem
- consider application of such simple systems and evolution in a game environment

Video - Algorithms and Evolution

Algorithms and evolution starts at 31:20 minutes into the video.

Source - Algorithms, YouTube

Games and formal elements

importance of objectives

- objectives may help establish different requirements and goals in a game
  - *helping a user to achieve results within the confines of the rules of the game*
- objectives may seem challenging and difficult
  - *need to be correctly designed relative to a game's rules*
  - *they should also seem achievable to a player*
- a game's objective may also help set the tone for gameplay and interaction
  - *e.g. objective of most platform games different from sports-based game*
  - *tone for each of these games becomes a reflection of the objective*
- use of objectives in games is not limited to just the overall game itself
- may consider defining an objective for different player roles
  - *or perhaps as mini challenges within our games*
- each level may define its own objective
  - *such as completing a level as fast as possible*
  - *collecting all available options (coins, for example) on another*
- choice of such objectives needs to be considered carefully
- each will affect not only the formal system of our game
  - *but also the dramatic aspect*
- good integration of objectives in the premise or story of a game
  - *helps strengthen dramatic aspects*
  - *e.g. Legend of Zelda*

Games and formal elements

a consideration of procedures

- we may start to see a few common actions that exist across multiple genres
- these often include the following:
  - *an action to start the game*
    - specific procedure required to initiate gameplay...
  - *ongoing actions and procedures*
    - e.g. common, persistent actions that continue, repeat &c. as part of the game
  - *reserved or special actions*
    - e.g. actions that may be required and executed due to a given condition or game requirement
  - *actions to conclude or resolve*
    - e.g. resolve actions at certain points within the game, or at the end of the game itself...
- for video games, incl. consoles, mobile, PC...
  - *such actions and procedures closely associated player interactions*
  - *e.g. given key combinations or controller buttons*
  - *perhaps tapping particular options on the screen itself*
  - *or moving a mobile device to control certain actions...*
- consider Super Mario Bros.
  - *we may clearly identify controls for given actions and procedures*
    - expected usage for directional buttons
    - option to jump or swim with the **A** button, &c.

Games and formal elements

procedures in development

- procedures also play a key role in the way we develop our games
- we can add procedures within the logic of our game
  - *to monitor certain ongoing states, user interaction, updates, rendering...*
- these procedures are working in the core of our game
  - *responding to changes in state*
- e.g. a player completes a puzzle within the main game
  - *need to monitor the ongoing puzzles responses*
  - *check the player input and interactions*
  - *then update the state of the game in response to a success or failure result*
- we're effectively checking whether a given action succeeds or not
  - *then, determine impact this may have on the game itself*
- such procedures and actions are naturally limited by real-world constraints
  - *e.g. performance of the underlying system, controllers, interaction options, screen...*
- may need to tailor such requirements to match the type of game we're developing
  - *and the target audience...*

Games and formal elements

rules and game concepts

- as we define and formalise rules for our games
  - *need to consider more than simply the gameplay itself*
- objects in games, and concepts embedded in gameplay structures
  - *require defined limitations and rules*
- game objects
  - *characters, weapons, vehicles, obstacles...*
  - *may be derived or inspired by real world objects*
- objects may come with the perception of existing limitations and rules
  - *a player knows what these objects can and cannot do in the real world*
- we may use these real world objects as inspiration
  - *starting points for our game's objects*
  - *not inherently limited or defined by them*
  - *may modify as befits the requirements of our game, and its gameplay*
- game context will be a determining factor in development of our objects
- objects may also be developed as a group of properties and variables
  - *together form the whole from varying requirements*
- in a world of chivalry, knights, ogres, and other fantastical creatures
  - *we may still create concepts and objects that unify these characters*
  - *from base objects, we can simply inherit and modify as needed*
- e.g. we may require various characters to ride
  - *on horseback, or perhaps astride an elephant, or even a fictional dragon &c.*
- our objects may be abstracted to include known attributes
  - *which can then be used as the parent*
  - *use for multiple real and imagined objects, characters within our game*

Games and formal elements

rules, objects, and updates...

- as developers and designers
  - *need to ensure a balance between maintaining game objects and variables*
  - *creating an intuitive update for our users*
- unlikely our player will want to keep a manual tally of such updates
  - *need to consider how we may allow them to quickly and easily intuit game objects*
- for example, we may need to
  - *maintain a running total of game objects, such as coins, lives, energy levels*
  - *correctly inform the player of any updates*
- a player should be able to quickly learn the nature of these objects
- if they're too difficult or complex
  - *need to consider how this affects our player's gaming experience*
- also need to ensure that there is sufficient isolation between different objects
- a player should be able to discern differences without too much effort or guesswork
- updates may also be influenced by known restrictions in the game's rules
  - *useful in many respects*
  - *e.g. relative to boundaries, objectives, and objects themselves*
- by establishing rules, e.g.
  - *to restrict objects and their attributes*
- rules help create a known scale for state within our game
- player has defined restrictions
  - *they know what they can and can't do*
  - *risk and reward is set in the game's logic and gameplay*

Python and Pygame - Game Example I

render text to a game window - intro

- drawing text to a game window in Pygame can become a repetitive process
  - *in particular, as part of each window update*
- we may abstract this underlying game requirement to a text output function

```
 # text output and render function - draw to game window
def textRender(surface, text, size, x, y):
    # specify font for text render
    ...
```

- start by specifying a surface where we need to draw the text
  - *plus text to render, its size, and coordinates relative to surface*
- need to specify a font for the text to be rendered
  - *reliant upon installed fonts for user's local system*
- use a font-match function with Pygame
  - *helps abstract specification of exact font to a relative name*

```
# specify font name to find
font_match = pygame.font.match_font('arial')
```

- Pygame will search local system for a font with the specified name
- use specified font to create an object for the font
  - *we need this object to render text in the game window*

```
# specify font for text render - uses found font and size of text
font = pygame.font.Font(font_match, size)
```

Python and Pygame - Game Example I

render text to a game window - text drawing

- text we'll be adding to the game window needs to be drawn
  - *drawn effectively pixel by pixel*
- Pygame calculates drawing for each pixel
  - *creates the specified text in the required font*
- start by specifying a surface to draw the required pixels for the text, e.g.

```
# surface for text pixels - TRUE = anti-aliased
text_surface = font.render(text, True, WHITE)
```

- we're specifying where to draw the text
  - *the text to draw to the game window*
  - *a boolean value for anti-aliasing of text*
  - *and the text colour*
- need to calculate a rectangle for placing the text surface, e.g.

```
# get rect for text surface rendering
text_rect = text_surface.get_rect()
```

- then specify where to position our text surface
  - *relative to defined `x` and `y` coordinates, e.g.*

```
# specify a relative location for text
text_rect.midtop = (x, y)
```

- text is then added to the surface using the standard `blit` function, e.g.

```
# add text surface to location of text rect
surface.blit(text_surface, text_rect)
```

Python and Pygame - Game Example 1

render text to a game window - text draw function

- overall text draw function is now as follows,

```
# text output and render function - draw to game window
def textRender(surface, text, size, x, y):
    # specify font for text render - uses found font and size of text
    font = pygame.font.Font(font_match, size)
    # surface for text pixels - TRUE = anti-aliased
    text_surface = font.render(text, True, WHITE)
    # get rect for text surface rendering
    text_rect = text_surface.get_rect()
    # specify a relative location for text
    text_rect.midtop = (x, y)
    # add text surface to location of text rect
    surface.blit(text_surface, text_rect)
```

- call this function whenever we need to render text to our game window
- in `draw` section of our game loop, now add the following call, e.g.

```
# draw text to game window - game score
textRender(window, str(game_score), 16, winWidth / 2, 10)
```

Python and Pygame - Game Example 1

render text to a game window - add a game score

- common example of rendering text in a game window
  - *simply output a running score for the player*
- start by adding an initial variable to record the player's score, e.g.

```
# initialise game score - default to 0
game_score = 0
```

- then allow a player to score points for each projectile collision on a mob object
  - *e.g. laser beam hit on an asteroid*
  - *fun to set variant points relative to size of mob object*
- if we use the radius of each mob object
  - *perform a quick calculation for each* `collision`
  - *work out points per asteroid, e.g.*

```
# calculate points relative to size of mob object
game_score += 40 - collision.radius
```

- relative to the recorded `collision`
  - *simply get the radius per hit mob object*
  - *then minus from a known starting value*

resources

- notes = drawing-text.pdf
- code
- drawingtext1.py (game example with score)
- drawingtext2.py (abstracted - simple rendered text)

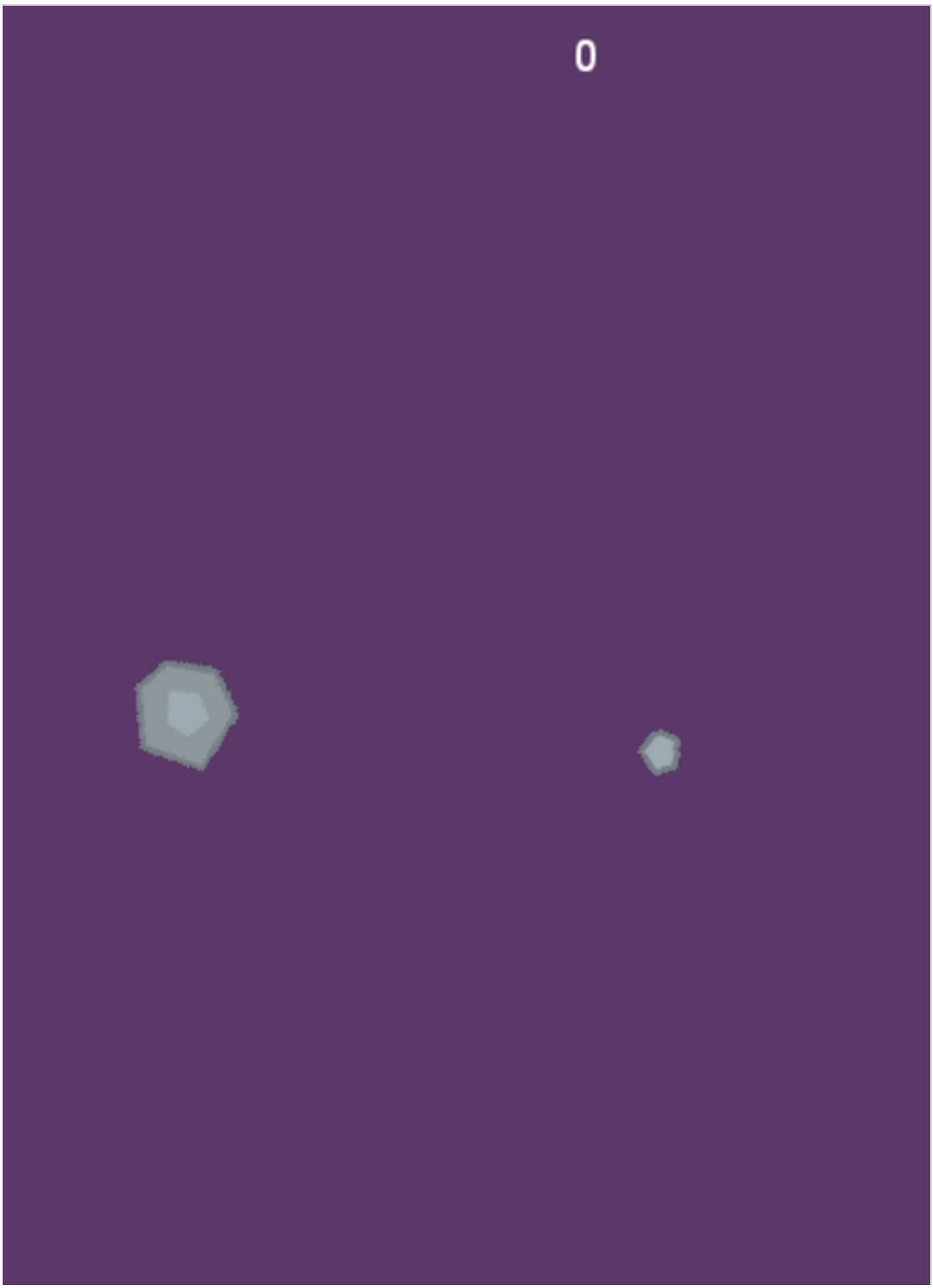
game example

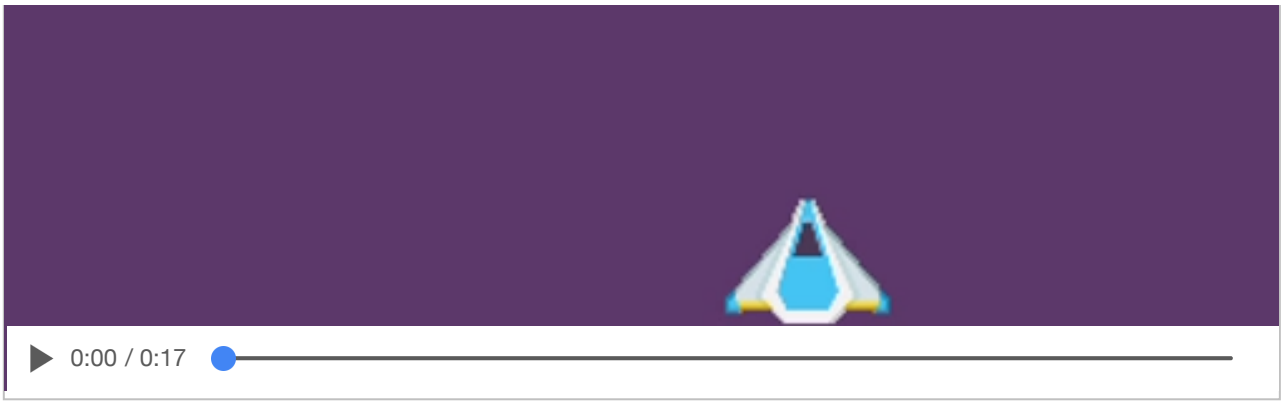
- shooter0.8.py

- draw text to the game window
- keep a running score for collisions with a projectile
  - *player shoots and destroys an asteroid*
  - *score is calculated relative to size of mob object - radius...*
- score is rendered to top of game window
  - *update for each successful hit*

Video - Shooter 0.8

render text for a game score




0:00 / 0:17

Python and Pygame - Game Example I

game music and sound effects - intro

- most of these sound effects will use a *WAV* format
  - *may also use other file formats such as OGG*
- add these files for our sound effects to the game assets directory, e.g.

```
|-- shootemup
    |-- assets
        |-- images
            |__ ship.png
        |-- sounds
            |__ laser-beam-med.wav
            |__ explosion-med.wav
```

Python and Pygame - Game Example I

game music and sound effects - import sounds and effects

- we need to add support for Pygame's `mixer`
  - *add the following call after we initialise Pygame itself, e.g.*

```
# add sound mixer to game
pygame.mixer.init()
```

- to use these sounds and effects in our game window
  - *need to add the directory location, e.g.*

```
# relative path to music and sound effects dir
snd_dir = os.path.join(assets_dir, "sounds")
```

- then start to add our required music and sound effects, e.g.

```
# load music and sound effects for use in game window
# laser beam firing sound effect
laser_effect = pygame.mixer.Sound(os.path.join(snd_dir, 'laser-beam-med.wav'))
# explosion sound effect
explosion_effect = pygame.mixer.Sound(os.path.join(snd_dir, 'explosion-med.wav'))
```

- add these lines of code right after we've loaded our images
  - *just before we start the game loop itself*

Python and Pygame - Game Example I

game music and sound effects - use sound effects

- after importing and loading our sound effects
  - *we may then choose where we need to play these sound effects in our game*
  - *e.g. player fires a laser beam to destroy falling mob objects*

```
# fire projectile from top of player sprite object
def fire(self):
    ...
    # play laser beam sound effect
    laser_effect.play()
```

- also add sound effects for each mob object explosion

```
# play laser beam sound effect
laser_effect.play()
```

Python and Pygame - Game Example 1

game music and sound effects - use music in a game

- as we add sound effects, we may also load music to play in the game
  - *we may add background music for the game window, e.g.*

```
# load music for background playback in game window
pygame.mixer.music.load(os.path.join(snd_dir, 'space-music-bg.ogg'))
```

- also set a relative volume for this background music
  - *creates ambience and does not overwhelm the player experience, e.g.*

```
# set music volume - half standard volume
pygame.mixer.music.set_volume(0.5)
```

resources

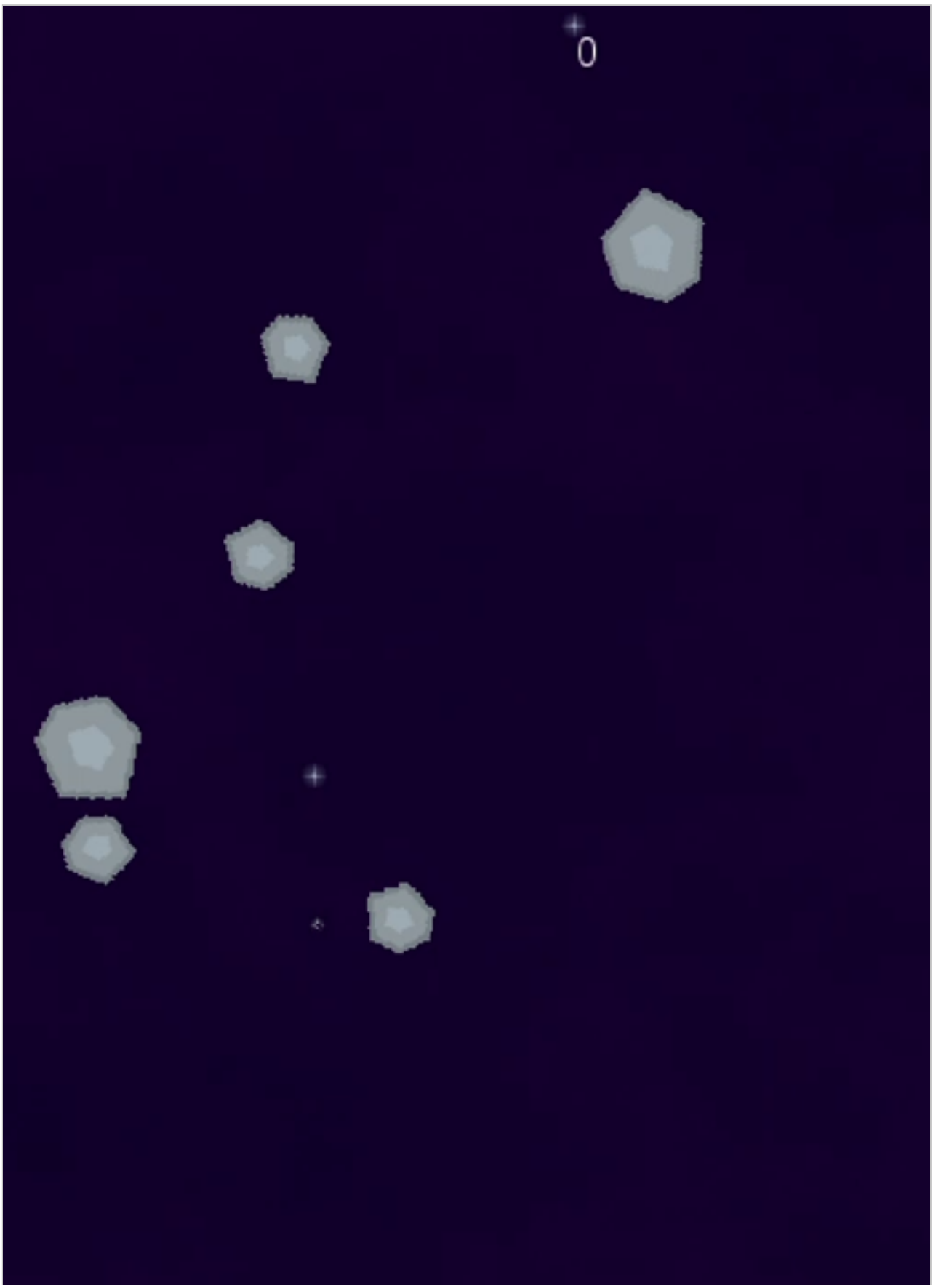
- notes = music-intro.pdf
- code
- basicmusic1.py
- basicmusic2.py

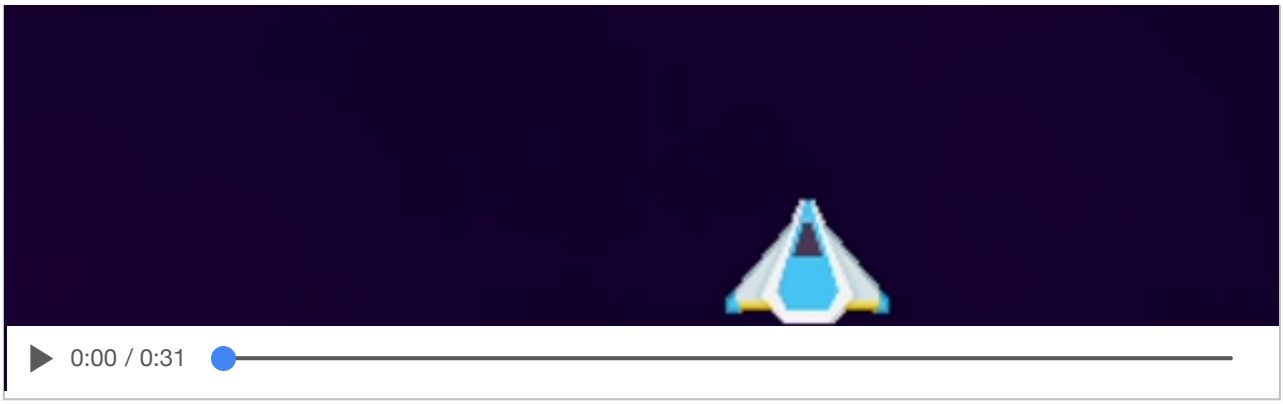
game example

- shooter0.9.py
- add music and sound effects to the game window
  - *add pygame mixer*
  - *load sounds directory in assets*
  - *load required sounds and sound effects*
  - *call `play()` for each required sound effect and game music...*

Video - Shooter 0.9

add music and sound effects




0:00 / 0:31

Python and Pygame - Game Example I

health and status - intro

- may add a status bar for a player's health, lives, &c.
- then dynamically update it relative to a defined health value
  - *e.g. a percentage value we decrement per collision*
- current game only gives a player one chance to shoot and destroy mob objects
  - *in effect, player currently has one life*
  - *one player life is not expected for most shooter style game...*
- may now consider monitoring and updating the status of a player's health
  - *e.g. as they are hit by advancing mob objects*
- protect our player, and their ship, using a Star Trek style shield
  - *may offer full protection initially*
  - *then incrementally weaken with each hit from a mob object*
  - *weakens until it eventually fails at value 0*
- set a default for this shield in the `Player` class,

```
# set default health for our player - start at max 100% and then decrease...
self.stShield = 100
```

Python and Pygame - Game Example I

health and status - collisions and shields

- need to modify our logic for a mob collision to ensure we handle such objects better
  - *may now reflect a decrease in the player's shield, health...*
- instead of allowing a mob object to continue after it has collided with the player
  - *now need to remove it from the game window*
- if we don't update this boolean to `True`
  - *each mob object will simply continue to hit the player*
  - *hit registered as it moves, pixel by pixel, through the player's ship*
  - *single hit would quickly become compounded in the gameplay*
- update for this check, e.g.

```
# add check for collision - enemy and player sprites (True = hit object is now deleted from game window)
collisions = pygame.sprite.spritecollide(player, mob_sprites, True, pygame.sprite.collide_circle)
```

- as our player may be hit by multiple mob objects
  - *also need to update our check from a simple conditional to a loop*
  - *check possible collisions...*

```
# check collisions with player's ship - decrease shield for each hit
for collision in collisions:
    # decrease player's shield for each collision
    player.stShield -= 20
    # check overall shield value - quit game if no shield
    if player.stShield <= 0:
        running = False
```

Python and Pygame - Game Example I

health and status - replace mob objects

- we still have an issue with losing mob objects
  - *if they collide with the player's ship*
  - *follows same underlying pattern as player's laser beam firing on mob objects*
- need to create a new object if it is removed after a collision
- a familiar pattern we may now abstract
  - *creation of mob objects to avoid repetition of code, e.g.*

```
# create a mob object
def createMob():
    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)
```

- simple abstracted function allows us to easily recreate our mob objects
  - *by creating a mob object*
  - *adding it to the overall group of* `game_sprites`
  - *then the specific group for the game's* `mob_sprites`
- then call this function if a mob object collides with a projectile, player's ship...
- also call this function when we initially create our new mob objects

```
# create a new mob object
createMob()
```

Python and Pygame - Game Example I

health and status - health status bar

- already defined a default maximum for our player's shield
  - *now start to output its value to the game window*
- we could simply output a numerical value
  - *as we did for the player's score*
- more interesting to show a graphical update for the status of a player's health
- define a new draw function to render a visual health bar for player's shield, e.g.

```
# draw a status bar for the player's health - percentage of health
def drawStatusBar(surface, x, y, health_pct):
    # defaults for status bar dimension
    BAR_WIDTH = 100
    BAR_HEIGHT = 10
    # use health as percentage to calculate fill for status bar
    bar_fill = (health / 100) * BAR_WIDTH
    # rectangles - outline of status bar &
    bar_rect = pygame.Rect(x, y, BAR_WIDTH, BAR_HEIGHT)
    fill_rect = pygame.Rect(x, y, bar_fill, BAR_HEIGHT)
    # draw health status bar to the game window - 1 specifies pixels for border width
    pygame.draw.rect(surface, GREEN, fill_rect)
    pygame.draw.rect(surface, WHITE, bar_rect, 1)
```

- function accepts four parameters, which allow us to define
  - *a surface for rendering*
  - *its x and y location in the game window*
  - *then update the status of the player's health*
- set a default width and height for the status bar
  - *then specify how much of this bar needs to be filled with colour*
  - *colour fill relative to the player's current health status...*
- health status can be calculated as a percentage

- *allows us to easily modify the relative sizes for the status bar*

resources

- notes = player-health-intro.pdf
- code = playerhealth1.py

Python and Pygame - Game Example 1

fun game extras - intro

- now start to add some fun extras to the general gameplay
  - *help improve the general player experience*
- a few examples
  - *modify health status bar to better reflect health percentages*
  - *auto fire for the laser beam to continuously shoot using space bar*
  - *fun explosions for collisions*
- many more...

Python and Pygame - Game Example 1

fun game extras - update health status colours

- modify health status bar to more accurately inform player of ship's health
- common option is to simply modify colour of status bar to reflect health status
- we may use a bright colour to indicate greater health status
  - *then change it to RED as a warning to the player, e.g.*

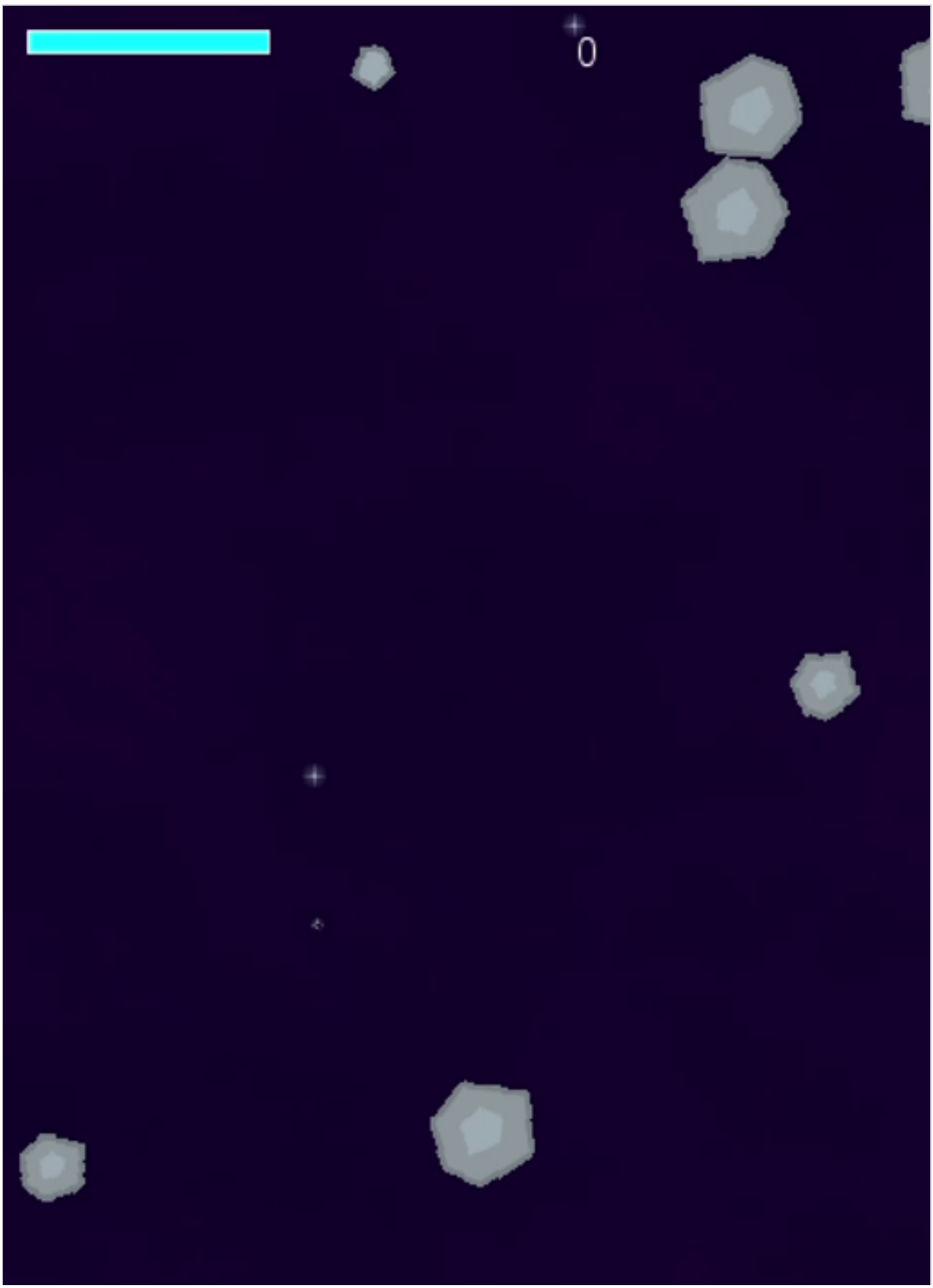
```
if bar_fill < 40:
    pygame.draw.rect(surface, RED, fill_rect)
else:
    pygame.draw.rect(surface, CYAN, fill_rect)
```

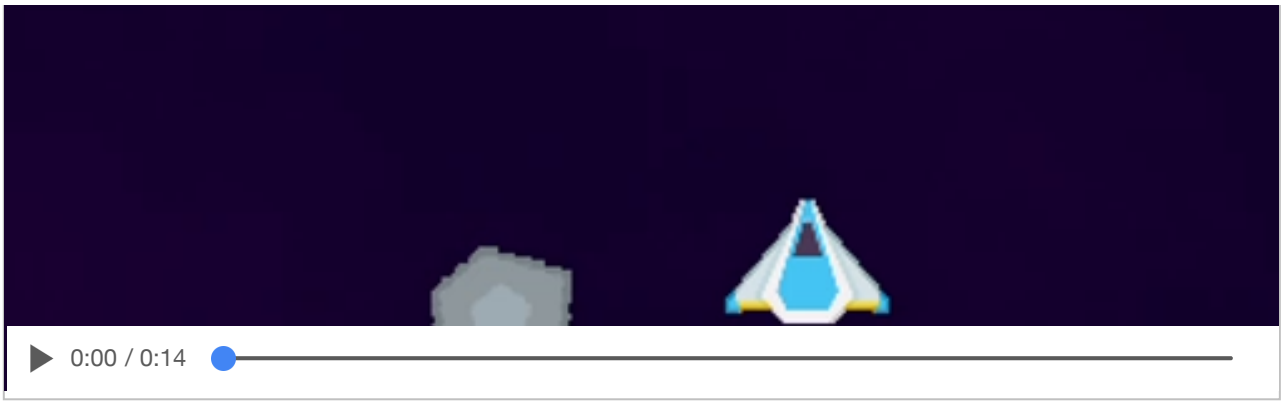
game example

- shooter1.0.py
- check player's health
  - *set default health to 100%*
  - *decrement health per collision*
    - quit game when health reaches 0
  - *draw status bar to game window*
    - green colour for good health
    - change to red colour below 40%

Video - Shooter 1.0

check player's health




0:00 / 0:14

Demos - Pygame

random sprites

- animatingsprites2.py

drawing text

- drawingtext1.py (game with text)
- drawingtext2.py (simple rendered text)

music and sound effects

- basicmusic1.py
- basicmusic2.py

health and status

- playerhealth1.py

Demos - Pygame - Game 1 Example

- shooter0.7.py
- shooter0.8.py
- shooter0.9.py
- shooter1.0.py

References

- Huizinga, J. *Homo Ludens: A Study of the Play-Element in Culture*. Angelico Press. 2016.
- Poundstone, W. *Prisoner's Dilemma.* Touchstone. New York. 2002.
- Salen, K. & Zimmerman, E. *Rules of Play: Game Design Fundamentals*. MIT Press. 2003.

References - Conway and Life Patterns

- LifeWiki
- Richard Guy
  - *Glider*

References - Pygame - Game Notes

- sprites-animating-random-images.pdf
- drawing-text.pdf
- music-intro.pdf
- player-health-intro.pdf

References - Various

- BFXR
- Homo Ludens
- Open Game Art
- SFXR

Videos

- Algorithms - YouTube