Integration

Saving

- First question: What to save?
  - The entire world state?
  - Selected objects?
- Comes down to design
  - I've not worked on a game that had to serialise the world
  - Much easier to work with a timeline
  - Saving flags is less error prone

Saving

- Important to ask yourself why you're serialising objects.
- Often see people do this at a very low level
  - Serialise the player
  - Serialise NPC states
  - Serialise states of doors, pickups etc.
  - Then attempt to load a scene, iterate over all these objects and deserialise from a file

Saving

- It's far simpler to save events
  - Has player found key?
  - Has player unlocked door?
  - Has player completed this quest?
- The state of our world is encapsulated by identifiable booleans
- Maps to our game state architecture
  - Awake() “If this has happened, what state do I need to be in? Do I even exist?”

Saving

- There's another big advantage
  - You can flip to any world state, in the editor
  - Pause the splash screen, tick off the events, let the level load
  - This can be defined in code
    - QA can jump to any “timeline”, repeatably
  - You can be sure that you're jumping to a point late in the game that maps exactly to what the world would be in natural play

Saving

- QA can share world state with the bug.
  - Dump the timeline, you can load it and jump to the bug location
- Code changes don't affect it, only design changes
  - “We no longer have this event”
  - In practice this hasn't ever bitten me...

Saving

- If you serialise a lot of objects
  - You can record points through natural play and share them
  - Code changes may break these shared saves
  - QA will find it harder to dump the world state
  - You may break player's save games if you update
- But it is at least easy to do this reliably in Unity now.

Saving

- Second question: what are the platform requirements?
  - Steam / console integration have their own rules
  - Enforce a large amount of rigour
- What about player use case?
  - Syncing?
  - Multiple save slots?
  - Multiple devices?

Saving

- Gets complicated quickly
- The player's crown jewels!
  - No matter what happens, the save game must survive!
- An open vector for attack
  - We've all done it… ;)
- What should happen when the game is uninstalled?

PlayerPrefs

- PlayerPrefs
  - In built way to save tuples
  - PlayerPrefs.SetString("Player Name", "Foobar");
  - Platform agnostic, same code everywhere
  - Uses the registry on windows!
  - Limited data-types; int, string, float
  - Meant for simple stuff life screen res, etc.

PlayerPrefs

- It is entirely possible to use this just to save a few scores
- Will persist if game is uninstalled
- Will not sync
- Will not get backed up
- Is plain text so can be easily modified
- This is more than likely fine for Uni projects...

Json

- Finally part of the standard API!
- Will allow low level serialisation of class instances marked [Serializable]
  - string json = JsonUtility.ToJson(myObject);
  - myObject = JsonUtility.FromJson<MyClass> (json);
- If you have a GameEvents class, with all your bool events in, job done...

Json

- Ish…
- Uses the half arsed Unity Serialiser
  - Dictionaries not supported
  - Nested Arrays not supported
  - Generics not supported
  - Only handles MonoBehaviour, ScriptableObject or plain class instances
  - Etc…
- Does this matter?

XML

- XML's a first class citizen in .Net so easy to use
  - using System.Xml;
  - using System.Xml.Xpath;
- It's not as hands off as the Json serilizer
  - You'll need to iterate over whatever you're saving
  - Create elements, Add Elements
- This is why most people hate it…
- Xpath is great, but no need for you to learn it...

Where?

- Where to save?
- Steam cloud has a couple of options
  - Auto-cloud: back up specific files from a predetermined location
    - Install Dir, MyDocuments, AppLocal, AppRoaming, etc
  - Full-cloud: you use the api to push the data to the cloud…
- Consoles work in a similar manner
- IOS/Android more custom

Where?

- Easy:
  - Create a save folder next to executable location
    - One of the Steam options, so hidden from the majority of players
  - Path.GetFullPath(".") + "/savegames";
  - DRM free versions will delete saves when uninstalled… GoG etc.
- Or:
  - Application.persistentDataPath
    - AppData\Local on windows ~\Library on Mac

Plain Text?

- I think we all know why not to do this…
- At a minimum, save as binary

```
BinaryFormatter bf = new BinaryFormatter();
Stream stream = File.Open(sPath, FileMode.Create);
bf.Serialize(stream, sMyString);
```

- For real fun:

```
byte[] keyArray = UTF8Encoding.UTF8.GetBytes("GARETHRULES");
byte[] toEncryptArray = UTF8Encoding.UTF8.GetBytes(sMyString);
RijndaelManaged rDel = new RijndaelManaged();
rDel.Key = keyArray;
rDel.Mode = CipherMode.ECB;
rDel.Padding = PaddingMode.PKCS7;
ICryptoTransform cTransform = rDel.CreateEncryptor();
byte[] resultArray = cTransform.TransformFinalBlock(toEncryptArray, 0, toEncryptArray.Length);
return Convert.ToBase64String(resultArray, 0, resultArray.Length)
```

Plain Text

- Binary formatting is still trivial to break
- Players WILL do this
  - And then expect you to support them when the game is broken.
- A bit of encryption gets you away from that
- But!
  - Release and Debug builds!

Other stuff...

- Steam API:
  - https://steamworks.github.io/
- Game sparks:
  - https://www.gamesparks.com/
- Don't roll your own!

Localisation

- Never ever ever ever put a raw string into the code!

```
public enum tLOC_Identifier
{
    _UI_FE_START = 1,
    _UI_FE_PLAYGAME = 2,
    _UI_FE_SETTINGS = 3,
    _UI_FE_EXITGAME = 4, etc...
```

Localisation

```
public static string[] LOC_English = {
   "Start",
   "Play Game",
   "Settings",
   "Exit Game", etc…

public static string[] LOC_Finnish = {
   "Aloita",
   "Pelaa",
   "Asetukset",
   "Poistu",etc...
```

Localisation

```
switch (m_gcGameEventsManager.m_iPREFS_Language)
{
  case GameGlobals.tLOC_Languages._English:
   sString = LOC_Strings.LOC_English[(int)iStringID];
   break;
  case GameGlobals.tLOC_Languages._Finnish:
   sString = LOC_Strings.LOC_Finnish[(int)iStringID];
   break;
```

Localisation

- The enum: `enum tLOC_Identifier` will be a dropdown list in the editor
- Anywhere you need a string, use the identifier!
- You can change languages by setting one variable.
  - Steam API: `GetCurrentGameLanguage();`

Localisation

- The strings can be auto generated
  - For Lumo I had:
    - Open Doc Spreadsheet
    - Localisation company can edit the spreadsheet
    - Python script that parsed the spreadsheet and generated the LOC_Strings.cs class
  - Any time there's a bug, update the spreadsheet, run the script.
  - Example in my Github:
    - TripleEh/Misc/...