Entity IDs And Communication

Forms of Entity Identification:	Pointers, Names, Entity UIDs (Evaluate these)
Entity Pointers(pros/cons)	Problems occur when entities are destroyed. For example when an entity high up in the chain dies, therefore, making the pointer invalid. Can lead to exceptions
Entity UIDs(Unique Identifier	+ Holds a UID instead of a pointer
	+ Uses a function to safely convert into pointer
	+ An error is returned when resource doesn't exist. A lot easier to handle than invalid pointers.
	+ Hash tables are used which provide efficency. Keys are converted into integers.
	- Collisions can occur in hash tables
Entity Messaging	Better to send messages to entities than to use getters and setters
	+ Entities only need to know messaging types
	+ Can make interactions more complex, by having replies for example.
	Need a system messenger class. Avoid using new and delete. Use statics

Entity IDs And Communication (cont)

- Finding the position of another entity can be clumsy

- There is latency between responses

+ Entity UIDs can be converted to pointers to access generic or commonly used data

+ Could implement an ImmediateMessage function where message is sent directly to entity and return value is response

Component-Based Entities

Problems with OO	Tight couping between parent and child. Features of parents affect or limit the features of children
	Hierarchies are static, games need more flexibility
	Multiple inheritance can be use but causes confusion
Component-based Architecture	Entity holds a dynamic list of components
	Each component has an update function called when entity is updated
	Messages to entity passed to each component. e.g. health component reacts to a damage message
	Send messages to components/entities within the same entity

Component-Based Entities (cont)

+Litte coupling between components

+Easy to add/remove functionality

+Simple to conceptualise

+Easily built from script/data files

-Much more message passing

-May be too flexible

Camera Projection/Picking

Model Space	Entity's mesh is defined in its own local coordinate system
World Space	Transforming a model in the world
World Matrix	Transforming model from model space to world space with a matrix.
Camera Space	The scene as view from the camera's position.
View Matrix	Transformation from world space to camera space is done with the view matrix.
Camera to Viewport space	Project camera space into 2D.
	This is done with the projection matrix
Projection Details	**Near clip distance** is from camera position to viewport.
	Far clip distance is furthest we can see from camera position.
	FOV - field of view

Tools Programming

Tool Chain	Sequence of tools needed to convert raw assets through to usable game data.

Influence Map

Influence Map	Way of viewing the distribution of control over a map.
	Grid out the world and provide a numerical estimate of the influence of every unit on the cell it is in and its neighhbouring cells.
	Influence diminishes over distance.
	The influence of all units in the game are summed in order to generate an influence map which is a representation of influence and location which can be used for strategic analysis.
	An example can be using the euclidean distance to calculate influence on each square/cell. influence = $0.5^{distance}$
	Calculation is done for every unit with positive values used for friendly units and negative for enemies.
Front line	Line that can be traces at the edge of positive and negative cells.
Concentration of forces	The areas with the highest positive values are where the influence of the friendly forces are strongest.

Influence Map (cont)

Maths	The influence falls off over distance which can be linear or exponential with distance. Influence is represented by the type of weapons for example. ie a sword is less powerful than a gun
	Because inflenuence never reaches 0, use a cut off point for very small values to avoid unnecessary calculations.
Desirability	A weighted sum which would change accordingly to the context or type of decision.
Time & distance	Even though influence diminishes over distance it can still have an impact on decisions taken by other units far away
Time & probability	Can suggest how we use influence maps to represent potential actions
Interpreting Results	Influence state decision such as combat want to choose acell in which enemy is weak but in which we are strong
	Examine the distance of units to the front-line both friendly and enemy which can help indicate areas to which we should be paying special attention to.

Influence Map (cont)

Terr ain	Can increase or decrease the propagation of influence according to terrain. e.g. take obstacles into account

Blackboard Model

Blackboard Model	Is a decision making method.
	Problems and all workings out are written on the blackboard.
	The insight is that a collective understanding of a problem may be better than an individual understanding.
	May be more efficient to have many experts each with a partial understading of a problem than one expert that has a full understanding.
Specialists	No specialist understands the whole problem
	Component that can operate on the data written p on the blackboard. The area of expertise of each specialist is narrow. A specialist may indicatate a relevance value indicating how they can deal with the problem.
	No communication allowed between experts. Everything goes through the medium of the blackboard.
Arbiter	Selects which of the specialists to execute
Architecture	2 types of architecture

Blackboard Model (cont)

	A) Multiple specialists each with their own area of expiertie. It is assumed that only one specialist at a time will be dealing with a specific problem.
	B) Specialists with overlapping areas of expertise. More than 1 specialist can signal relevance. More than one specialist can deal with a problem at a time.
Characteristics	Blackboard offers flexibility
	Order of reasoning not pre-determined
	At any given point that most relevant specialist will be selected
	Specialists can act in a variety of ways like requesting more data etc.
	A specialist need not know how its assertions or signals are going to be used.
	The specialist is only concerned with fulfilling a request.

Production Systems

Knowledge Representation	Knowledge is representation and the methods for manipulating it
Procedural Knowledge	Is operational ie what to do when
	Most common method is production rules
Production Rules	New knowledge is derived using various reasoning mechanisms. IF AND THEN

Production Systems (cont)

A deductive argument can only bring out what is already implicit in it premises but can give rise to questions.
Reasoning is carried out by an interpreter
2 methods: forward chaining from assertions and backward chaining from hypotheses

Trees

Decision Trees	A decision tree is way of representing knowledge.
	A decision tree is a way of using inputs to predict future outputs.
	It is a classification method, decision trees learn from examples using inductions. They can deal with uncertainty. thye dont use ranges because of large numbers of branching alternatives.
Sequence	Execute the first node that has not yet succeeded. Keep executing a task until it returns a success. Any failure in the sequence is a failure overall. Fix or continue to next sequence.
Selector	Selects one child node to execute. Could be random or some sort of control mechanism.
Decorator	Single child node. Allows for other types of operation such as repetition, filter or an inventor.
ID3 algorithms	Note: Look at Tree PP for method

Entity Update and Rendering

Entity Update	Each entity has its own update function
	Can be called every frame or less frequently
	Recieves/processes messages
	Send messages
	Decision making
	+ Entity behaviour and state collected together
	+Easy to maintain
	+ Easier to comprehend behaviour
	- Overall game behaviour is distributed, can get unexpected interaction
	- Messaging between entities can be long-winded
Scene Update	In the TL-Engine a single global update function is used
	Can become bloated/hard to maintain
	No attempt to encapsulate behaviour
	Using entity-based update still uses Scene Update to do certain global update work
	+ Global function easy to work with
	- Model state tends to become a set of globals
Entity Rendering	Each entity gets its own render function. A function that's called every frame to update animations, positions, textures etc.
Pre/Post Rendering	Called for entities before and after the main rendering calls. E.g. Calculating camera view matrix

Entity Update and Rendering (cont)

Dot Product Formula	X.T = |X||T|cos(B)

Text-based Game Data

Hard-coding	Embedding of data in program code
	- Requires recompilation to change data which can be slow for large project.
	- Cannot be done at runtime
	To improve this we can use data files.
	+ Hard coded data is stored in a text format which means it is human readable/writable.
	Text based data will need to be parsed at run-time.
Binary data files	Can help in large data sets as it's quicker to parse but they're not human readable
Issues with Text-based data	Slower than using hard-coding
	Text need more storage than binary
	Need good test cases to have good text validation.
	Additional code development required
XML (eXtensible Markup Language	Structured data
	Not a programming language
	Stream-oriented parsing - Uses callbacks as tags that are opened and closed

Text-based Game Data (cont)

	Tree-traversal parsing - Reads entire document and passes back as a complete hierarchical structure
XML Disadvantages	The redundancy of syntax causes higher storage making parsing take longer.
	XML is less readable compared to other text-based document formats such as JSON. XML doesn't support arrays.
	XML files are usually larger due to being verbose therefore it totally depends on who is writing it.
Advantages XML	XML is platform and programming language independent therfore can be used by any system and supports hardware and software change.
	Support Unicode and international encoding standard for use with different languages and scripts.
	The data stored and transported using XML can be changed at any point of time without affecting the data presented. XML allows validation using DTD and Schema. This validation ensures that the XML document is free from any syntax error.

Text-based Game Data (cont)

XML simplifiesdata sharing between various systems because of its platform independent nature.

Concurrent Programming

Concurrent Program	Simultaneously executes multiple interacting computational tasks. Not the same as parallel program
Processes/threads	The tasks may be separate programs or a set of processes/threads created by a single program.
	Focus of concurrent programming is the interaction between tasks and the coordination of shared resoruces.
Parallel Programming	Simultaneously exutes a single task across several processors
Processes	A program is just a passive set of instuctions whereas a process is an active instance of a program, acually being executed
	Each process has a distinct set of resources. A section of memory (RAM, cache). System Resources. Security settings (perms), processor state
Threads	A program may in turn contain several threads of execution
	Threads conatin process resources Look Above ^^
	Processes can be single threaded or multi-threaded

Concurrent Programming (cont)

	Multi-threaded can be more efficent if done right
	Multi-threaded processes are more efficient than multi--process programs due to less setup and communication since threads share resources.
Data Coordination	Major issue with concurrent is preventing concurrent processes from interfering with eachother.
Resource Coordination	Preventing sharing of resources from interfering. e.g. One process rewrites content of the file while another is in the process of reading it.
Race Conditions	2 processes racing to complete their task first
	A flaw in a concurrent system where the exact sequence or timing of events affects the output.
	Hard to track down due to shared data/resources being accesses almost simultaneously.
Locking	A resource, piece of data or section of code can be locked to a single process or thread.
Critical Section	A section of code that can only be accessed by a single thread at a time.. The section of code is aasumed to be accesing data that needs careful synchronisation. Only locks code not data.

Concurrent Programming (cont)

Mutex	An object that can only be owned by a single thread on a single process at a time.
Semaphore	An obkect that can be held by up to N threads simultaneously. Section can be shared by a few processes but not an unlimited number, which limits the number resources that can be opened simultaneously.
Timers	Can pause a thread until a certain time or repeatedly wake/sleep a thread.
Blocking	When a thread or process is prevented from accessing data or executing code due to synchronisation object is said to be blocked.
	When a thread is blocked wait for the code/data to become available by allowing the thread to stall (sleep) , which loses the advantages of concurrency. And can add a timeout to help limit how long to wait. Or simply skip the task that requires the blocked data/code

Concurrent Programming (cont)

Dead locks	When 2 threads try to lock 2 resources they stall waiting for eachother causing a deadlock and each thread will wait forever for the other. Can only be resoved by better synchronisation of objects. ie associate a single mutex to the ownership of any part of the group.

Planning

STRIPS (Stanford Research Institute Problem Solver)

- Formal language that assumes that all conditions not stated to be true are false
- Planning is a process of divising a sequence of actions to achieve a goal.
- Pathfinding is an example of planning
- Uses actions, states and goals
- In language can be expressed as logical statements like At(B). They can be combined like At(door) AND holding(key)
- Actions can be specified in terms of preconditions. Like Move(A, B), Preconditions: At(A), Postconditions: not At(A), At(B)
- Precondition = entry state
- Postcondition = exit state

Finite State Machines (FSM)

FSMS model states, transitions and actions

Probablistic FSM	Describe any FSM which includes probabilities
	Probabilities are placed on transitions out of states

Finite State Machines (FSM) (cont)

	Can have an output state which has a probability associated with it.
	Multiple output states with probability scores used to select between them
	Probabilities could be fixed or could change over time. Can extend probabilities in lots of ways e.g. trigger functions.
Stack-(-based FSM)	Track past states using a stack
	Stacks are pushed on and popped off the stack at transitions. This means that an agent can be interrupted and later return to a previous state.
	Stack based FSM can produce a simple FSM than a standard FSM but not always appropriate tor return to a previous state.
Hieracrchical FSM	A state may link to another FSM or set of FSMs
	Transition from a state leads to a brand new FSM. Use the stack to store the initiating state.
	If control is passed down the hierarchy then the new FSM starts at its own initial state. Allow you to identify and separate out behaviour or tasks. Helps reduce size and complexity of a FSM
	Record the original state and any associated data because control may pass back at some point.

Finite State Machines (FSM) (cont)

	- May lead to code re-use since a task could be used in several different situations
	To avoid code repitition allow the re-use of FSMs.
	The hierarchy of states can produce behaviour unique to an agent even if states are shared with other agents.
	It is possible to swap FSMs in and out.
	This can be done with any one of the FSM layers.
	Hence an agent could exhibit different implementations of a task in different situations, e.g. different combat FSMs.
	A state could have sub states
	This can bypass the need to have a new FSM but avoid doing it too much or else it can lead to the FSM becoming broken.
Subsumption FSM	Intelligent behaviour can be built from a collection of simple machines.
	Decompose complex behaviour into simple modules, operations or tasks.
	The modules are implemented as layers of FSMs
	Thje layers of FSMs all operate at the same time.
	Lower layers deal with short-term goals and higher layers deal with long-term goals.
	Lower layers have priority

Resource Management

Resource/Asset	Any file that is loaded and used by elements in the game
Asset Management	PRogramming involved in loading and working with asset files
Resource Template	Template that stores the information about the assets in the game.
Resource loading issues	System automatically loads all the level resources at setup time. No hard coding
	Repeatition of resource loading. Therefore, need to identify resources that have already loaded.
	Could load resources on demand when entity is needed
Shared Resources	Find if resource has already been loaded. Can serach the entire list but may be slow
	Use hash map instead for efficency. Could use UIDs like with entities.
Resource Destruction	Could destroy all objects at the end of level
	Could destroy exlicitly so each entity has a delete function.
Track Resources	Track resources and delete them when they're not being used.
Smart Pointers	Pointer that manages its own memory and atomatically detect the reference count which is increased/decreased according to the reference count.

Resource Management (cont)

Reference count issues	If reference count reaches 0 they are deleted and may need to be used later on
	Reloading can cause stutter in game, which we want to avoid.
	To deal with this issue we can store a single persistent reference throughout the game.

Scripting for Games

Why Scripting	Ease of development - Less prone to erros and less intricate
	Much easier to change and test
	No recompilation, change at runtime
	Think about Unity - Use scripts to control entities(game objects) player as an example
Scripting (pros/-cons)	- Performance - Scriping language often interpreted. Can be 10x slower than C++
	- No control of memory management can cause issues
	- Limited tool support
	- Hard to spot errors
	- Need to write interface to our C++
	Don't neccessarly need to use scripting languages
	Consider language based on performance needs and memory footprint, feature set etc.
Python	Portable, interpreted, OO programming language

Scripting for Games (cont)

	Dynamically typed
	Automatic garbage collection
	Blocks are defined by indentation
Lua	Lightweight scrupting language
	Not OO
	Small/Simple feature set
	Small memory
	Small but powerful feature set
	Dynamically typed
	Only one kind of data structure - the table
	Simple integration with C API
	Less high level than Python
	Rather niche language outside games
	Better performance, less memory use
	Simple interface for C and C++
	Lends itself well to game entity scripting
	Interfacing LUa with C++ is fairly simple since Lua is itself a C program and has a direct C API.

Cellular Automata

Cellular Automata	Are machines which model problems as a set of discrete cells.
Game of Life	John Conway
	Uses a 2D grid as a map to lay out the actions of the game.

Cellular Automata (cont)

	Binary cells used to represent entities on the map with either alive or empty where empty is dead.
	Each cell only considers its 8 neighbouring cells: orthogonal and diagonal
	All cells are examined simultaneously
	Each cell considered in its own right.
Game of life rules	A live cell with less than two live neighbours dies. Analogous to loneliness or underpopulation. A live cell with more than three live neighbours dies. Analogous to overpoulation or crowding. A live cell with two or three live neighbours survives. It becomes part of the next generation of cells. An empty cell with three neighbours becomes a live cell.
	Need to seed the system with alive cells to start the game otherwise nothing happens.

Cellular Automata (cont)

Rules	1. A live cell with less than two live neighbours dies. Analogous to loneliness or underpopulation. A live cell with more than three live neighbours dies. Analogous to overpoulation or crowding. A live cell with two or three live neighbours survives. It becomes part of the next generation of cells. An empty cell with three neighbours becomes a live cell.
	2. A live cell with more than three live neighbours dies. Analogous to overpoulation or crowding.
	3. A live cell with two or three live neighbours survives. It becomes part of the next generation of cells.
	4. An empty cell with three neighbours becomes a live cell.
	Refer to lecture powerpoint for game of life examples

Terrain Analysis

Applicability	Wide variety of approaches. From Team based games, squads, enemy AI, moving into cover, adopting to a good firing position etc.
Specific Requirements for terrain analysis	Representation of terrain
	Reason about that representation

Terrain Analysis (cont)

	Difficult to generalise
	Typically custom built
	General points can be made
Initial Analysis	Need to decide the attributes being used in the reasoning. Cannot recognise a choke point unless you hae already decided that these are of use to your game.
Waypoints	Reasoning using waypoints
	Need a representation of the world.
	For each waypoint calculate its offensive and defensive value
	Directional information needed
	Can take various factors into consideration including cover, lack of target etc.
Static and Dynamic - Preprocessing	Some static analysis is comparatively easy. Hills shore etc.
	This can be pre-processed
	More difficult with dynamic terrain though
Clustering	A strategory game needs to be able to recognise dynamic areas like towns and forests.

Terrain Analysis (cont)

	The region is complex. Better to convert into convex hull.
Convex Hulls	Easy to reason with.
	eed to know what points are inside the convex hull.
Choke points	Use an influence that can grow.Each region is surrounded by a uniform area. Arny areas that overlap are considered to be choke points.
	Choke points can be extended to show where to hide. This is done by tracing along the edge of a region going away from the choke point until there is no direct line of sight to the choke point.
	Influence maps have been used for terrain analysis to identify locations such as resource points, building routes for attack or staging areas for attack.

Terrain Analysis (cont)

Cover behind objects	Simplest case is single opponent firing at you and you track a line of sight to the edges of the object. Any point in between the two edge points is in cover. Can be used for multiple opponents. Perfect location for cover can be calculated by calculating the centre of gravity of the object. Assume that the object is 2D and that mass is evenly distributed.
	Simply trace a line from centre to oppoent or opponents

Turing Machine

Turing Aproach	Turing defined a class of abstract machines now called Turing Machines
	Turing is breaking maths down to its most basic operations.
Turing Machine	Recasts this idea as a machine he supposes can perform all of the functions that the man does.
	Turing defined a class of of abstract machines now called Turing Machines.
	A mathematical model of computation that defines an abstract machine which manipulates symbols on a strip of tape according to a table of rows.

Turing Machine (cont)

Relevance to computers	Turing machines can do recursions, add and do functions. You can create any mathematical operation we know about using these basic operations.
Universal TM	A basic TM can compute only one particular function. Where Universal TM is one which can simulate any other machine.
Turing's Thesis	The definition of computation is "something which can be done by TM".
Church	Demonstrated that any computation can be done using Lambda calculus.
Issues with TM	The halting problem: the determination of whether a TM will come to a halt given a particular program. Disproof by showing a contraction. It posits the existence of a program to solve the Halting Problem and then demonstrates that it would lead to a contradiction.
Proof	Testing proves in general halting problem cannot be solved. The reason is that it gives rise to an inherent contradiction.
Humans	Human minds might be Universal TMS as it has been argued that a Universal TM should in principle be capable of intelligence.

Turing Machine (cont)

Real compu ters	Universal TM is comparable to real computer. Anything that a real computer can compute a TM can compute. It is easier to describe certain algorithms using a TM than using a real computer.
	Universal TM are unbounded with infinite space, where computers are bounded both time and space are limited. TM express algorithms in general terms where as a real computer needs to consider other things such as precision and error conditions.
	A TM uses a sequential tape. A real computer uses registers and random access storage. TMs do not model concurrency easily i.e. different tasks performing at the same time.