20220405-软件架构&算法导论

1.过程描述

1.1 Software architecture

1)4+1view model

Logical view：

- focused mostly on achieving the **functional requirements** of a system.The context is the services that should be provided to end user
- defining all of the **classes**; their attributes; their behaviors
- showing the **relationship** between **software objects and components**
- **State diagram** and **classes disgram** are most commonly used

Process view

- focus on achieving nonfunctional requirements which specify the desired qualities for the system, including quality attributes like performance and availability

- show the **execution order** of different objects, and the calls to methods defined by the logical view in the correct order. Behaviors that are asynchronous or concurrent are also described
- **UML sequence diafram** and **UML activity diagram** are often used

Development view

- consider things like **programming languages, libraries and tool sets**
- also includes management details like scheduling, budgets and work assignments. Especially project management

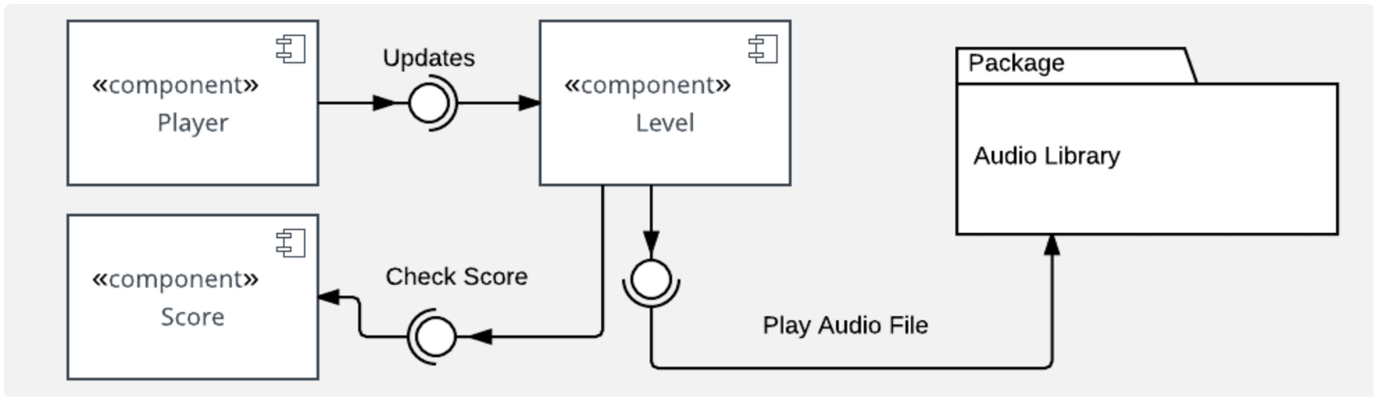
Physical view

- handles how elements in the logical process and development view to be mapped to different **nodes** of **hardware** for running the system.
- **UML deployment diagram** can express

Scenerios

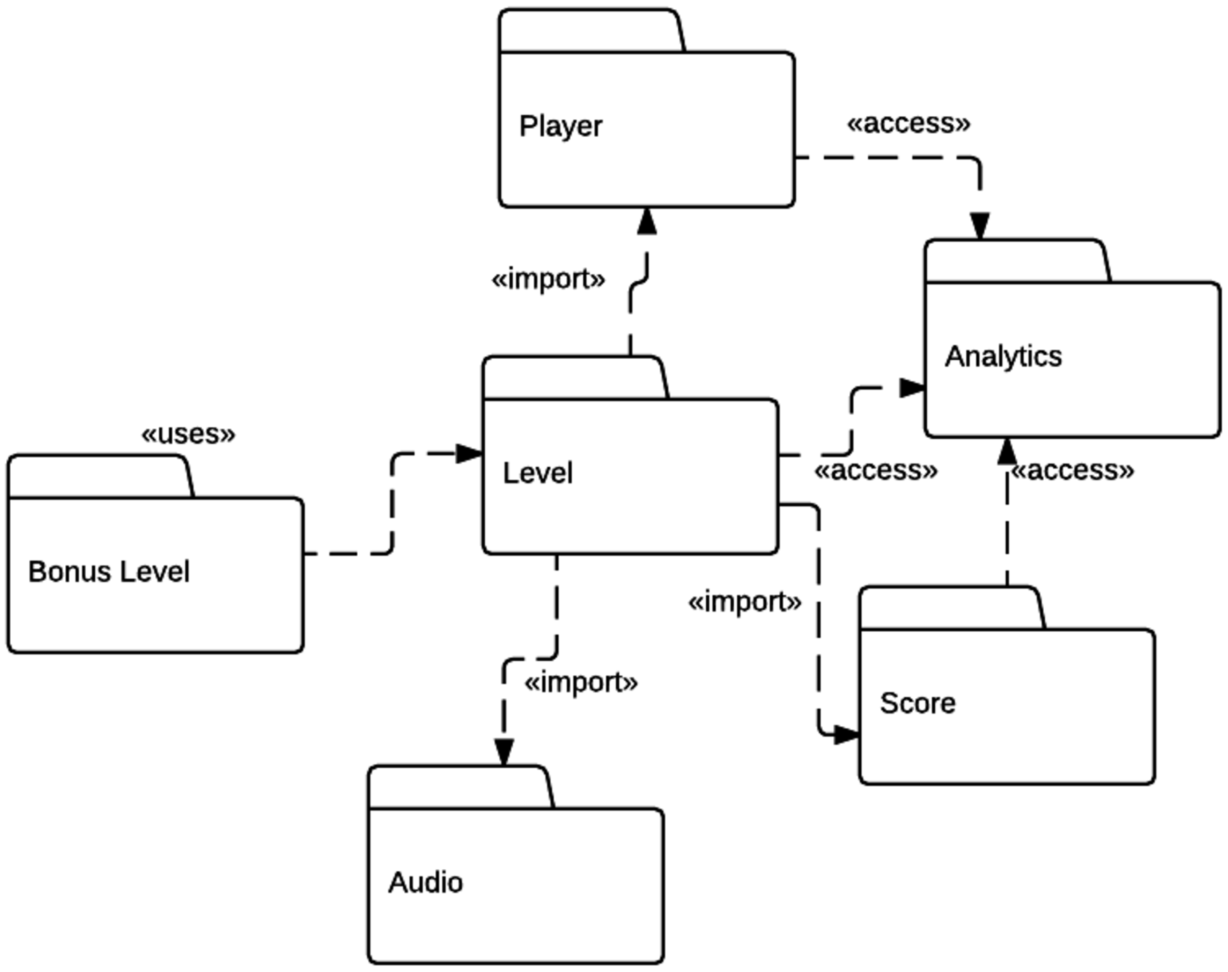
- align with the use cases or user tasks of a system
- how the four other views work together
- often use script that describes **the sequence of interactions between objects and processes**

2)Component diagram



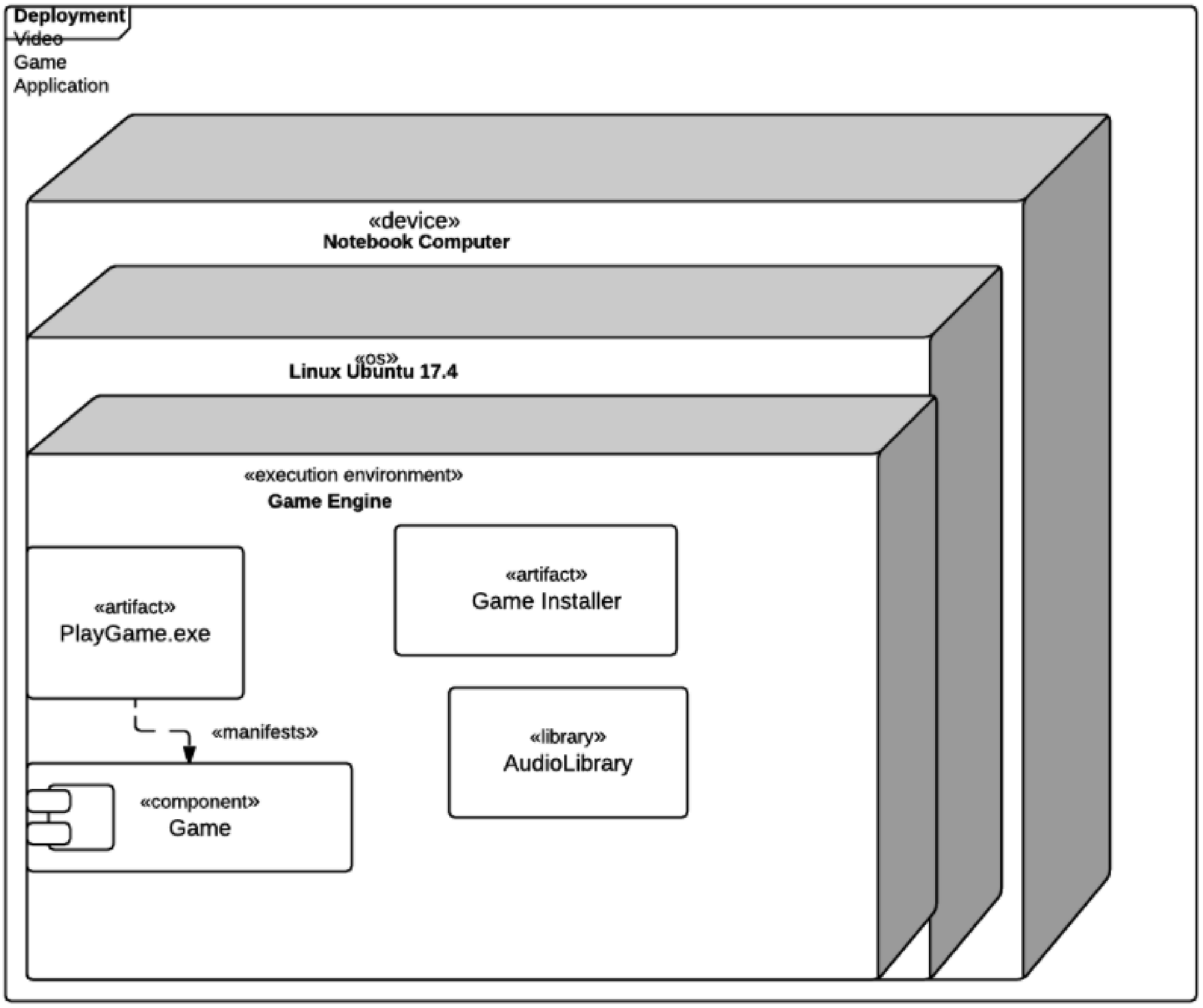
- the basis are component and their relationships
- a ball connector, is how you display a provided interface in component diagrams. the purpose of a provided interface is to show that **a component offers an interface for others to interact with it**
- a socket connector, display a required interface. to show that **a component expects a certain interface**
- first indentify the **main objects** used in the system, then indentify all of the **relevant libraries** needed for the system

3)UML Package diagram



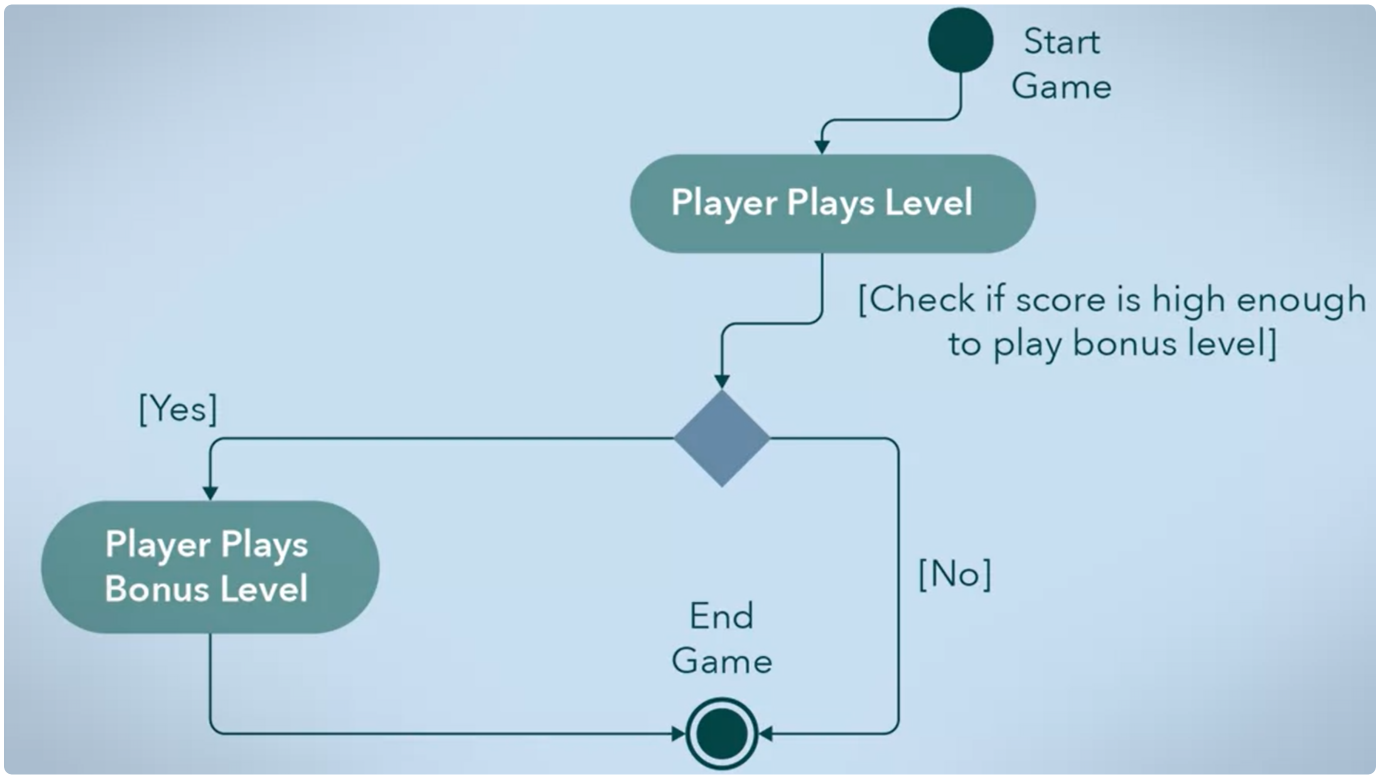
- A package **groups together elements** of your software that are related based on data, classes or user tasks. Also defines a **namespace** for the elements it contains
- Package diagram shows **packages and the dependencies** between them
- In the package, certain elements can be marked as public or private based on the + and – signs in front of their names
- The **import** tag says the import is public, the **access** tag says the import is private, no need to make names in the imported further known outside the namespace

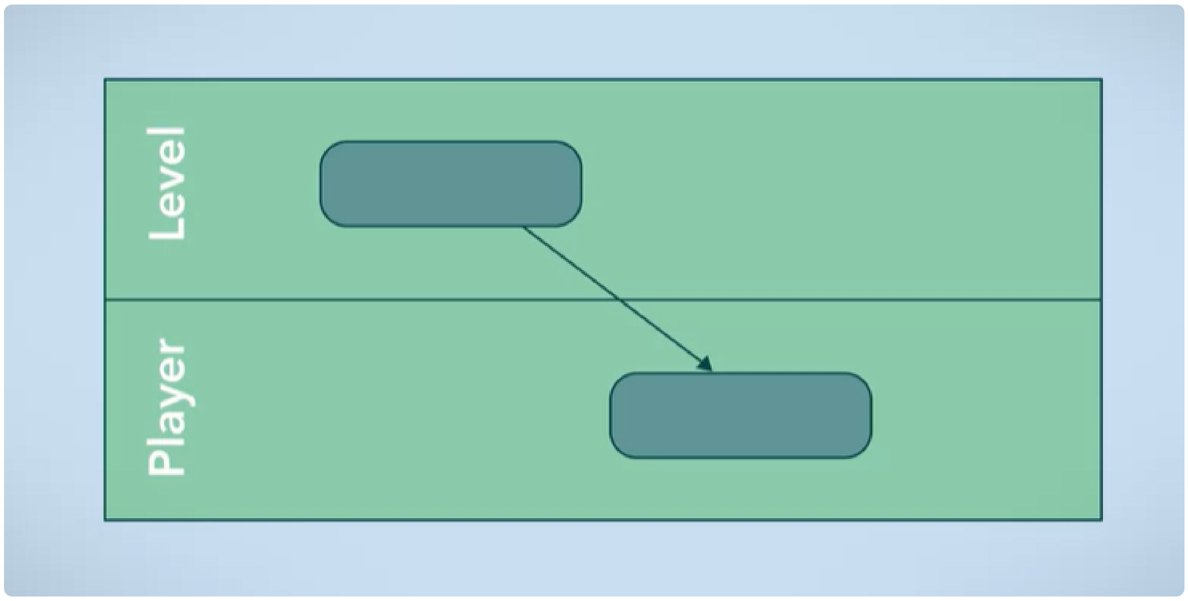
4)UML Deployment diagram



- The deployment digram gives a high level look at the **artifacts, libraries, main components, machines and devices** that your application needs to run
- **Artifact** is a physical result of the development process, final pieces to put together
- **specification level diagram** gives an overview of artifacts and deployment targets, without referencing specific details like machine names
- **instance level diagram** is a much more specific approach, which map a specific artifact to a specific deployment target, and can indentify specific machines and hardware devices
- **Nodes(the cube)** are deployment targets that contain artifacts available for execution. Hardware devices are also displayed in the same way as nodes
- Manifestation is where an artifact is a **physical realization of a software component**

5)UML Activity diagram



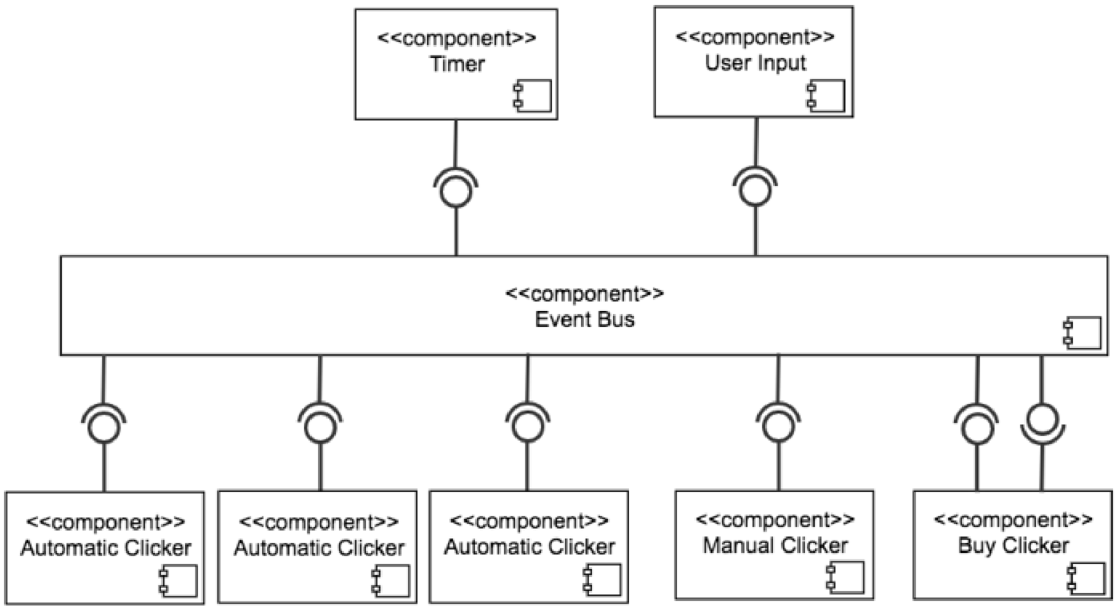


- In the activity diagram, you represent the **flow** from one activity to another in a software system. The purpose is to capture the dynamic behavior of the system, how control flows from one activity to another
- **activities** and **conditions** are two essiential parts
- there can be **parallel flows**
- **Partition** divide activities up into different catefories, such as where it occurs or the user role involved. **Swim lanes** are used to dislplay these partitions

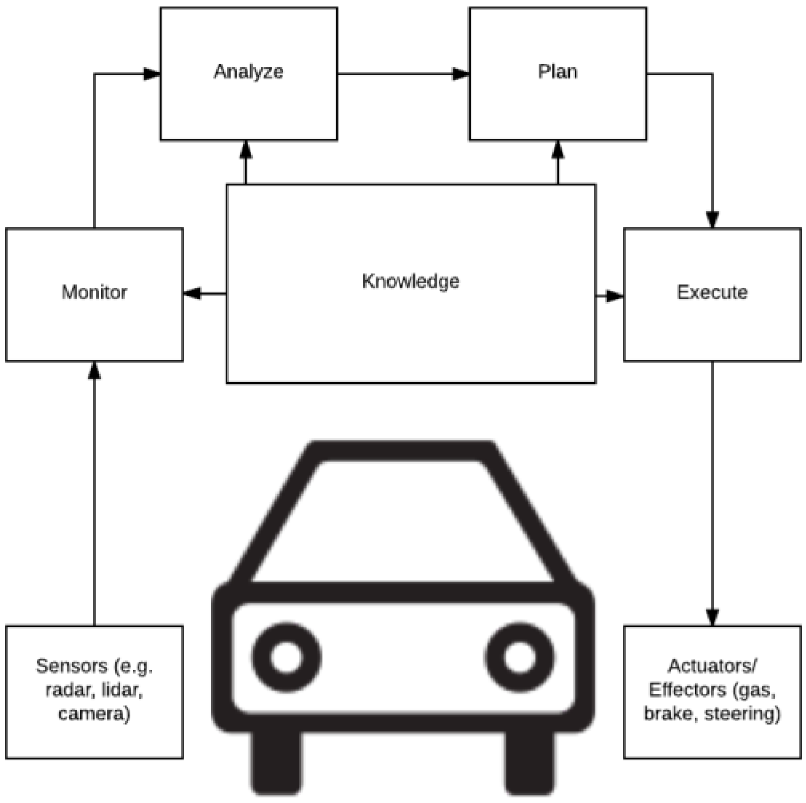
6)Architectural styles

Name	Definition	Keypoints
languag e–based systems	types: • abstract data types and object–oriented architectural style • Main program and subroutine	inherience is allowed in object–oriented sytle
repositor y–based systems	data–centric, allow data to be stored and shared between multiple components. Can be achieved by integrating a method of shared storage such as a databse	
layered systems	layer is a collection of components that work together towards a common purpose. Components only interact with components in their own layer or adjacent layers	• top layer have no authority to allocate system resources or communicate with hardware directly. This "sandboxing" provides security and reliability to the kernel • loosely coupled, follows the principle of least konwledge
Client Server n–tier	tiers refer to components that are typically on different physical machines. relationship between 2 adjacent tiers is often a client/server relationship	• A tier can be both server and client • request–response relationships can be synchronous or aysnchronous • requires extra resources to manage the client/server relationships

Interpreter–based systems	allow end users to write scripts, macros or rules that access, compose and run the basic features of those systems. Encourage customization	portable java is a example
Dataflow systems	pipes and filters architectural style. perform transformation on data	
Implicit invocation systems	event–based architectural style. Events are both indicators of change in the system and triggers to functions. Events can be signal, user inputs, messages and etc	• functions take the form of event generators(send events) and event consumers(receive and process events). a funcation can be both • communication between functions is mediated by an event bus, so it is called implicit invocation • each event consumer registers with the event bus to notified of certain events • when the bus(always listening) detect an event, it distributes the event to consumers • 2 consumers can running at the same time on shared data. Semaphore is used to coordinated this process. Semaphore is a variable or abstract data type that toggles between 2 values.
Process control systems	feedback loops with sensor, controller, actuator and the process itself	



<<component>> Timer
<<component>> User Input
<<component>> Event Bus
<<component>> Automatic Clicker
<<component>> Automatic Clicker
<<component>> Automatic Clicker
<<component>> Manual Clicker
<<component>> Buy Clicker



Analyze
Plan
Monitor
Knowledge
Execute
Sensors (e.g. radar, lidar, camera)
Actuators/ Effectors (gas, brake, steering)

2.结果输出

今天看了软件架构课程的第二个模块，主要讲了一些典型的架构类型，感觉看下来收获不是特别多，就当是练练英语了。晚上决定之后一段时间主要完成算法导论的阅读，之前规划的确实不太合理，也很难

真正掌握知识。趁着现在因为疫情动弹不得，就耐着性子把英文原书好好读一下。