

2025 年北航敏捷软工结对项目：影蛇舞

前言

欢迎来参加 2025 年北航计算机学院敏捷软工的结对编程项目！🎉

结对编程项目的设计，意在为大家提供相对陌生的问题背景和技术场景，从而促使大家在相对集中的工作时间内和伙伴一起活用各自的经验和知识，完成一次结对编程、极限编程的体验。

任务的构成

需要完成的任务

编程实现

⚠️ 请在开始结对编程项目之前，**clone/fork** 本项目代码仓库。你需要确保你的代码仓库在项目 **DDL** 前保持不公开（或：不将 **commits** 推送到公开的远端仓库），在项目 **DDL** 后公开（或：将 **commits** 推送到公开的远端仓库）。

你可以根据网络环境，选择 [GitHub 仓库](#) 或者 [GitCode 仓库](#)。

在接下来的任务描述中，会出现标注为类似 🧑 **T1**（依序编号）的编程任务，每一个编程任务会对应本代码仓库下的一个文件夹，例如 `/T1` 对应代码实现任务 **T1** 的内容。

相关编码实现请在本仓库的基础上进行。

对问题的作答

在接下来的任务描述中，会出现标注为类似 📖 **Q1.1(P/I)**（依序编号，可能含子任务）的问题。

标记为 **(P)** 的问题，需要两个人经过相互讨论给出共同的答案：体现在提交的博客中，相关内容两个人的答案相同。

标记为 **(I)** 的问题，需要两个人经过独立思考给出各自的答案：体现在提交的博客中，相关内容两个人的答案不同。

请依照任务指导在完成结对编程项目的同时，作答各项问题。

问题列表位于：仓库的 `question.md` 文件，请不要将本文档内的作答提交到代码仓库。

需要提交的项目

代码仓库

每组（两人）只需提交一份，仓库至少在课程结束前开源在主流代码托管平台（例如 GitHub、Gitee、GitCode 等）。

博客文档

每组（两人）需要各自提交一份，并发表在课程 `cnblogs` 社区。博客的内容即为对应 `question.md` 文件两人各自的内容。

项目的预想和要求

限时完成

本项目设计上是一个限时完成的项目，课程组对项目的预想总完成时间（包括编码、编码时简单作答的时间，不包括思考、后续依个人情况完善作答等难以界定的时间）为 **12 小时**。希望大家尽量确保连贯、成块的时间完成本结对编程项目，单次保证至少 **1~2 小时** 的投入，推荐尽量一次至少完成一个章节。

因此，对每项任务，会提供课程组的预期实现时间供参考；同时，会要求记录每项任务的开始时间和结束时间，相关时间记录应当和代码仓库的 `commit` 记录相对应。

如果中间有所中断，请诚实地记录每个分段的起止时间。

对时间的要求并不会影响得分，大家如实记载即可。一种情况例外：由于本项目包含对各位的代码实现进行“比较”、确定“性能分”的部分，有意采用远远超出时限的大工作量来取得更好性能的工作不会获得合理工时工作以上的得分。

结对完成

本项目应当以结对编程的形式完成。具体来说，期望大家采取两人线下（或线上）的方式，在一台机器上，一边讨论、一边工作，一位成员担当“驾驶员”、另一位成员担当“导航员”。

❌ 请不要采用对任务进行分解、两位成员互不沟通而各自完成某一部分的方式来完成本项目。

体验优先

本项目是重视体验、而非重视结果的设计，除去占比 30% 的性能分部分，均不涉及完成结果之间的对比。余下分数以编程实现完成既定目标为主，未能达标也会根据大家的体验情况和项目投入给出较为合理（一般不低于相关任务满分 60%）的评分——所谓大家的“投入”，是指在博客/问题作答中体现出的结对过程。

课程评价的组成

本项目的设计总分为 150 分。

项目	分数占比	预计耗时
博客文档（对各问题的回答）	50 分	/
引入 Guide: WebAssembly（无编程任务）	0 分	60 min
编程任务第一部分（主需求一：不涉及性能分）	30 分	120 min
编程任务第二部分（主需求二：不涉及性能分）	30 分	180 min
编程任务第三部分（主需求三：涉及性能分）	40 分	360 min

那么，请和你的搭档坐在一起，开始吧！

Guide: WebAssembly

Chapter.0 Belua multorum es capitums.（你是多首的怪物。）

 本章内容的参考完成时间为 60 分钟。

如何在 Web 场景中，为多种编程语言提供统一的运行环境而尽可能地利用其语言设计本身、工具链和生态环境所带来的效能优势？

——噫噫咚！突然问这样的问题实在是很生硬的引入，一般而言谁会去想啦！但是这里是极限编程的结对项目，请和你的搭档一起让大脑活跃起来——之后会一直保持这样的节奏！准备好 `question.md`，在代码仓库外复制一份，请边阅读边填写入代码仓库外的版本，或采取简记、语音备忘等方式记载较复杂问题的要点之后再补充。那么首先——

→  **Q0.1(P)** 请记录下目前的时间。

Quid faciam?（我该如何是好？）

回到一开始的问题，或许大家很容易会想到“😡 聊这个我可就不困了！JS，我讨厌你！”JS 作为 Web 开发的某种意义上的标准语言，在以其敏捷、多范式等等设计特点创建并不断发展 Web 生态的同时，也在通过标准化、编程语言本身及编译器和运行时的发展而不断演进。Web 应用的开发范式高速迭代，和软件工程领域的最前端紧密相连——时至今日，Web 浏览器引擎本身就已成为一套统一的运行环境和交互接口：从 ChromeOS 到小程序、再到诸如 Typora、VSCode、QQ、2023 年敏捷软工团队项目 Ficus 等等各种各样基于 Electron 的桌面应用……似乎从某一天开始，万事万物都可以“运行在浏览器里”。

然而！这并不像听起来那样美好得理所应当——即便 Google、Mozilla、Apple 等等企业和来自开源社区的力量日复一日地使用各类手段来优化其 JS 运行时的性能，似乎我们还是比想象中更快地到达了瓶颈。JS 语言以及 Web 生态的既有特性使得 Web 开发环境不成比例地倾斜向开发效率而非执行效率，而 Web 场景新的边界正将对性能的企盼以及那些沉痾痼疾不断地向外扩散——人们开始要求在浏览器里运行一段复杂的图形学算法、一个神经网络模型、又或者一个紧凑的模拟器。

等等！你当然会想“那倒是去像我们美好的过去一样，下载原生（Native）构建的软件、安装到自己的电脑上运行啊！”从开发者的角度来想，很容易观察到不同执行环境带来的额外适配成本、跨平台软件构建的开销复杂化了开发和分发，舍弃 Web 开发生态的诸多范式（例如，使用 Web 前端框架构建用户交互界面在跨平台场景下相较去和不同平台各自为政的 UI Kit 打交道，前者往往能更简便地实现统一的用户体验）更带来了许多头疼的问题；从用户的角度来想，能够通过网络分发的软件服务却要多执行一次和操作环境息息相关的安装过程，继而面临许多纷至沓来的问题——无论是使用一次还是使用多次、无论是在公用的设备还是私有的设备上……怎么想都很完蛋！Web 对软件开发及使用范式的改变如同打开的潘多拉魔盒，已经回不去了！

——Web 基础设施面对着诸多方面的需要，在各种各样的设计取舍中打转。面对这样多首的怪物，应该如何是好呢？

Facta fugis, facienda petis.（舍既成之本，逐未竟之业。）

要是能结合 Web 和 Native 的优点就好了！

那么，要让 Web 生态、或者说一种雄心勃勃的软件运行环境的未来，受益于既有的‘原生’软件开发模式、技术栈和工具链，我们还缺少什么呢？

由 W3C 及 Mozilla、Microsoft、Google、Apple、Fastly、Intel、Red Hat 等企业提供一种答案是，通过定义一套二进制指令格式（基本上可以说：一套字节码设计规范）和对应的基于栈式虚拟机的运行时设计规格、以及相关基础设施及生态工具链的设计协议来提供一种 Web 原生的解决方案。

这便是本项目（技术上）的主角——**WebAssembly** 🎉！

开源社区、企业——尤其是以 Web 生态相关企业为中心的字节码联盟（Bytecode Alliance）进而根据这些方案去实现相关的工具链，包括提供运行时、开发工具链的实现、完成对上下游的适配等等。

也许还是有些云里雾里！简单来说，就是定义一套综合考虑多种编程语言设计的、较低层级的统一字节码，上游对接各类编程语言——实现一套编译器后端将各类语言编译为 WebAssembly（缩短一点，Wasm！）字节码，下游对接目前的 Web 运行时——实现浏览器引擎的 Wasm 执行能力、或者更激进一些实现脱离浏览器的独立 Wasm 运行时。这样的好处显而易见——我们可以在 Web 生态中整合多种编程语言、生态系统、开发工具和开发范式的优势了！

eg. C/C++ 语言和工具链能够适应某些更性能敏感任务的开发需要，那么我们就用 C/C++ 来实现，然后编译到 Wasm 并在既有的（主要基于 JS 的）代码库基础上通过调用 Wasm 运行时来使用它，进而得益于这种性能提升。

得益于一套统一的基础，对更多编程语言及其生态的支持也就得以花费相对小的成本进行。在这样一套新的基础设施上，开发人员能够跳出 JS 的性能泥淖，但又可以保持在既有的生态中，实现渐进的转移。由此，我们得到了 Web 崭新的能力底座！

哪怕是 JS 语言本身，也因此获得了发挥长处而不是继续不堪重负的好去处。让合适的语言去合适的地方吧！在合适的应用场景，如性能不敏感的程序逻辑描述、处理各程序模块交互的胶水代码等领域，开发人员依然可以充分利用 JS 的优势。

在本次的结对项目中，Wasm 将站在我们的舞台中心！

→ 📖 **Q0.2(I)** 【你可以在结对结束后另行补充。】作为本项目的调查：

请如实标注在开始项目之前对 Wasm 的熟悉程度分级，可以的话请细化具体情况。（分别回答两人各自的情况）

I. 没有听说过；

II. 仅限于听说过相关名词；

III. 听说过，且有一定了解；

IV. 听说过，且使用 Wasm 实际进行过开发（即便是玩具项目的开发）。

WasM Dream! 啊喂没事吗？！（担心着也期待着）

接下来的任务需要你和你的搭档：

1. 使用任意语言编程实现列出的需求；
2. 使用相关工具链将代码编译到 Wasm；
3. 使用相关工具链生成或手工编写 JS 胶水代码，导出封装好的 Wasm 中你实现的函数；
4. 调试代码，确保你的实现能够对接课程组提供的测试用 JS 代码，这些代码将在 **Node.js** 的当前 **LTS** 版本 (**now at v22.14.0**) 运行时上运行。

由于目前支持 Wasm 作为编译目标的编程语言及工具链种类繁多，你可以参照 [Wasm 官方文档](#) 选择支持的语言进行尝试。代码实现只要能通过课程组提供的提交测试 JS 代码，都可以接受作为提交。然而，为了降低实践难度，课程组为以下两种编程语言提供额外的支持，为 C++ 提供更为有限的支持。你的选择对项目进行的影响如下表所示：

所选用的编程语言	课程组支持情况	对任务的影响
AssemblyScript	支持：提供资料，有限答疑（资料范围内）	任务设计的难度基线
Rust	支持：提供资料，有限答疑（资料范围内）	任务设计的难度基线
C/C++	部分支持：提供更为有限的资料，不提供任何答疑	由于工具链、语言和运行时设计等原因，完成任务的难度相对会更高，请谨慎选择
能够完成需求的其他编程语言	不提供参考资料支持，需要自行了解和学习	挑战性的：需要自行解决可能遇到的问题

请注意，对编程语言及工具链的选择，可能影响程序实现的性能表现。

课程组在课程社区提供了[指引](#)来帮助大家在规定时间内入门 WebAssembly，请各位根据自己的情况选择合适的编程语言。相关代码已位于代码仓库 [/G](#) 内。

阅读完成 Guide 后——

→  **Q0.3(P)** 请记录下目前的时间。

结对项目：影蛇舞

Chapter.1 不畏迷茫，只管前进。（迷子でもいい、前へ進め。）

 本章内容的参考完成时间为 120 分钟。

→  **Q1.1(P)** 请记录下目前的时间。

→  **Q1.2(P)** 请在完成任务的同时记录，并在完成任务后整理完善：

1. 浏览任务要求，参照 附录 A：基于 **PSP 2.1** 修改的 **PSP** 表格，估计任务预计耗时；
2. 完成编程任务期间，依次做了什么（比如查阅了什么资料，随后如何进行了开发，遇到了什么问题，又通过什么方式解决）；

雾周途

弥漫不散的浓雾中

传来呼唤我的声音 引领我前进

在这个属于蛇的年份里，丛林的深处出现了可爱的小蛇。

不知从哪儿来、不知往哪儿去——我们的小蛇沉眠许久，直到肚子饿扁扁！睁开眼睛，从遥远的山林的尽头传来了祖训：

“所谓蛇蛇们存活下去的意义，就是长期素食啊！和我、吃一辈子果子吧——”

小蛇充满信心地睁开了眼睛——

接下来将说明本结对项目的**基本规则**。

这些规则与贪吃蛇，*snake.io*，或者某手游里的某小游戏有一些相似之处。

不过需要澄清的是，我们对一些规则进行了规约和简化，所以请仔细阅读后续所有规则描述。

后续任务的规则，都将基于以下基础规则。

“脚下的路”，等等、蛇有脚吗？（各实体的定义）

• 场地

- 场地的形状是正方形，场地是由 8×8 个空方格构成的区域，类似国际象棋棋盘。我们将采用 $(1, 1)$ 到 $(8, 8)$ 的坐标表示所有的方格。坐标轴采用大家中学数学常用的方向，也就是右边是 $+x$ 方向，上边是 $+y$ 方向。
- 正方形的场地外是地形边界、也即场地边缘，场地边缘不能被穿过，场地外什么也不能出现。

• 蛇

- 蛇由四节身体构成。每个身体占据一个方格，两两相邻，互不重合。
- 蛇可以吃果子，蛇用脑袋吃果子！蛇的第一节身体是蛇头，蛇能且只能用头吃果子。但是和一般的贪吃蛇不同，因为你的蛇一直处在运动过程中，根据热量守恒定律，你的蛇的长度不会变长。
- 每个回合中，蛇头移动到和前一刻蛇头所在的位置相邻的一个格子上，蛇头后的每节身体移动到上一刻前一节身体的位置。
- 蛇会死！如果蛇的头撞到任何东西上（比方说边界、蛇身——除了果子），蛇就会死。一定要想办法避免你的蛇死掉！你不会有操作死蛇的机会。蛇死身消，整条蛇会原地蒸发。

• 果子

- 果子会在行动之前出现在棋盘空格上。
- 在本章中，场地里永远有 1 个果子，开始一局新游戏时或每当果子被蛇吃掉时，新的果子会出现在随机空格上。

“蹒跚的步履”（行动和时序）

每局游戏将按照以下方式推进：

游戏开始

在游戏开始时，蛇被添加到场地上的某个位置，果子被生成到场地上的某个位置。

游戏进行

1. **决策**：你可以看到此时棋盘的样子，并决定蛇的移动方向。蛇的移动方向可能为上（+y方向）、左（-x方向）、下（-y方向）、右（+x方向）。给蛇蛇一个指令吧！
2. **移动**：蛇会按照该方向移动。蛇的头（第一节身体）移动到某个相邻的格子上，同时蛇头后的每节身体移动到上一刻前一节身体的位置。
3. **判定**：判定此时蛇头的位置，蛇可能发生了碰撞或吃到了果子：
 - a. **碰撞**：蛇会撞死在蛇的身体或场地边缘上。具体条件是：蛇的第一节身体（即蛇头）和蛇的任何一节身体重合，或者在场地以外（即与场地边界重合）。
 - b. **得分**：如果蛇吃到了果子，本局游戏得分增加 **1 分**。
4. **补充**：回合结束前，缺失的果子被补充。
5. **收尾**：回合结束，除非满足了游戏结束条件，回到阶段 1。

游戏结束

满足下列游戏结束条件之一时，游戏结束。

1. 到达回合数上限，本章中为 200 回合；
2. 很遗憾，蛇蛇不幸离开了我们，被大家永远地怀念了（蛇死了）。

如果你还有疑惑的话，可以在课程群与助教确认规则。

果一会

若是能给予这果子一个归处的话

我便愿意前往

→ 🐍 **T1 贪吃蛇 1：吃果子！**

请根据以下要求及参考资料，完成任务。

本任务的全部源代码应该存放于 `/T1`。

代码有进展即进行 `commit`，`commit log` 不合理的项目会被助教抽查询问项目细节。

已经看清了这片丛林，就开始我们的蛇蛇觅食之旅吧！

要求


你需要设计一个函数决定蛇的移动方式。


- 函数名：`greedy_snake_move()`、`greedySnakeMove()`，或参考你所用语言命名风格选择合适的命名。
- 参数：为上述规则中所介绍的，“你看到的场地”的样子，由两个 `int32[]` 数组构成，依次为：
 - 四个蛇身坐标：每个坐标由两个 `int32` 构成，第一元为横坐标，第二元为纵坐标。数组的前两项为蛇头的坐标，之后依次是第二节身子、第三节身子、蛇尾（即第四节身子）的坐标。数组长度为 8。
 - 果子的坐标：由两个 `int32` 构成，第一元为横坐标，第二元为纵坐标。数组长度为 2。
- 返回值：一个 `int32`，代表蛇本轮选择的移动方向。
 - 输出值为 **0, 1, 2, 3** 四个数字中的一个，分别对应上（**y**）、左（**-x**）、下（**-y**）、右（**y**）四个方向。


在回合限制（最多 **200** 回合）和时间限制（最多 **1000 ms**）内，你设计的函数会被重复执行，直到你的蛇将果子吃掉。

测试

自行测试

你和你的搭档需要参照  **Guide** 中提到的测试方法，设计测试用例、实现测试代码并完成测试。这是一个必做的计分项目，会根据大家的测试设计和实现给予相应的分数。

→  **Q1.3(P)** 请说明针对该任务，你们设计和实现测试的方法及过程，包括但不限于：出于对需求的哪些考虑设计了哪些测试用例、如何评估所设计测试的有效性 等等。

→  **Q1.4(I)** 请说明单元测试对软件开发的作用。

提交测试

当你完成全部代码编写、测试任务后，请切换到目录 `/T1`；参照注释，根据你选择的语言/实现方式修改 `test.js` 开头的相关引入代码，只允许修改和你的选择相关的那一行，其他代码不允许修改！然后，执行以下命令来完成提交前的测试：

```
npm run submit-test
```

如果测试输出：

```
🎉 You have passed all the tests provided.
```

说明你的代码实现已被接受作为有效提交。

评价

满分 **30 分**。在 🏆 **T1** 中，你的目标是控制蛇每局吃掉 1 个果子。在 🏆 **T1** 中，只要你完成了这个目标，本局游戏结束。由于只需要 1 分，补充果子等事件并不会在这一阶段触发。你可以暂时不考虑这些规则，完成必要的部分即可。

不要使用复杂度过高的算法！你在一局游戏内的总耗时上限为 **1000 ms**，超过则会失去这一局的分数。

课程组会通过设计相关测试用例，对大家程序实现的正确性进行评价：即这些评测占本部分得分的一部分，按照通过的测试用例数占测试用例总数的比例给予相同比例的分数。

→ 📖 **Q1.5(P)** 请记录下目前的时间，并根据实际情况填写 附录 A：基于 **PSP 2.1** 修改的 **PSP** 表格 的“实际耗时”栏目。

→ 📖 **Q1.6(I)** 请写下本部分的心得体会。

Chapter.2 即使迷茫，也要前行。（迷子でもいい、迷子でも進め。）

🕒 本章内容的参考完成时间为 180 分钟。

→ 📖 **Q2.1(P)** 请记录下目前的时间。

→ 📖 **Q2.2(P)** 请在完成任务的同时记录，并在完成任务后整理完善：

1. 浏览任务要求，参照附录 A：基于 PSP 2.1 修改的 PSP 表格，估计任务预计耗时；
2. 完成编程任务期间，依次做了什么（比如查阅了什么资料，随后如何进行了开发，遇到了什么问题，又通过什么方式解决）；

孤坏牢

（无法容忍的）这些条条框框

是束缚不了我的心的啊

蛇蛇继续向前，可是障碍横在了眼前。远远看上去是草叶，扭扭曲曲地接近了才看清是木制的巨像——木头人。

初生的小蛇没有办法和木头人硬碰硬！只能把它们当作前行路上的障碍，绕过去，寻得吃到果子的未来。

接下来将说明关于障碍物的新增规则。

这些规则在  T1 的基础上增加了以下内容，仅适用于本章任务。

撒在路上的标记（新增的实体定义）

- 障碍物
 - 场地上共有 12 个障碍物。
 - 每个障碍物的大小均为一格。
 - 障碍物不会移动。
 - 障碍物不能被穿过，即蛇的头撞在障碍物上时会导致死亡。
- 果子
 - 由于障碍物的引入，在一局游戏中，果子可能被障碍物和边界阻隔而不可达。

使人陷入迷茫的公式（新增的时序规则）

游戏开始

在游戏开始时，蛇被添加到场地上的某个位置，果子被生成到场地上的某个位置。障碍物被生成到场地上的某个位置。它们互不重合。

游戏进行

3. 判定：判定此时蛇头的位置，蛇可能发生了碰撞或吃到了果子：

- a. 碰撞：蛇会撞死在蛇的身体或场地边缘、障碍物上。具体条件是：蛇的第一节身体和蛇的任何一节身体重合，或者在场地以外（即与场地边界重合），或者与障碍物重合。

其他规则不变。

如果你还有疑惑的话，可以在课程群与助教确认规则。

无路矢

纵使如入无尽深渊 始终无法收到回应

仍会持续呼唤你（果子）

→ 🤖 **T2 贪吃蛇 2：避障吃果子！**

请根据以下要求及参考资料，完成任务。

本任务的全部源代码应该存放于 `/T2`。

代码有进展即进行 `commit`，`commit log` 不合理的项目会被助教抽查询问项目细节。



我们的蛇蛇怎样绕过障碍物，如愿以偿地饱餐呢——


要求

你需要设计一个函数决定蛇的移动方式。

- 函数名: `greedy_snake_move_barriers()`、`greedySnakeMoveBarriers()`，或参考你所用语言命名风格选择合适的命名。
- 参数: 为上述规则中所介绍的，“你看到的场地”的样子，由三个 `int32[]` 数组构成，依次为：
 - 四个蛇身坐标: 每个坐标由两个 `int32` 构成，第一元为横坐标，第二元为纵坐标。数组的前两项为蛇头的坐标，之后依次是第二节身子、第三节身子、蛇尾（即第四节身子）的坐标。数组长度为 8。
 - 果子的坐标: 由两个 `int32` 构成，第一元为横坐标，第二元为纵坐标。数组长度为 2。
 - 障碍的坐标: 每个坐标由两个 `int32` 构成，第一元为横坐标，第二元为纵坐标。共有 12 个障碍物，数组长度为 24。
- 返回值: 一个 `int32`，代表蛇本轮选择的移动方向或不可达标记。
 - 若可达，输出值为 **0, 1, 2, 3** 四个数字中的一个，分别对应上（y）、左（-x）、下（-y）、右（y）四个方向。
 - 若不可达，直接输出 **-1**。

在回合限制（最多 **200** 回合）和时间限制（最多 **1s**）内，若可达，你设计的函数会被重复执行，直到你的蛇将果子吃掉；若不可达，你设计的函数应该在第一回合直接输出 **-1**。

→  **Q2.3(P)** 请说明针对该任务，你们对  **T1** 中已实现的代码进行了哪些复用和修改。

→  **Q2.4(I)** 请说明在编码实现时，可以采取哪些设计思想、考虑哪些设计冗余，来提高既存代码适应需求变更的能力。

测试

提交测试

当你完成全部代码编写、测试任务后，请切换到目录 `/T2`；参照注释，根据你选择的语言/实现方式修改 `test.js` 开头的相关引入代码，只允许修改和你的选择相关的那一行，其他代码不允许修改！然后，执行以下命令来完成提交前的测试：

```
npm run submit-test
```


如果测试输出：

 You have passed all the tests provided.

说明你的代码实现已被接受作为有效提交。


评价


满分 30 分。


课程组会通过设计相关测试用例，对大家程序实现的正确性进行评价：即这些评测占本部分得分的一部分，按照通过的测试用例数占测试用例总数的比例给予相同比例的分。

→  **Q2.5(P)** 头脑风暴环节：


只吃一个食物可满足不了贪吃蛇的欲望，请一起思考并简述以下场景中贪吃蛇的策略：

在  T2 的基础上，场地里不再是只有 1 个果子，而是总共有 n 个果子 ($1 < n < 10$)，果子随机分布在场中且不会刷新，保证不与障碍物重叠，保证每个果子均可达，且至少存在一条成功吃掉所有果子的路线，其余条件保持不变，请你找出一条吃完所有果子的行动路径。

→  **Q2.6(P)** 请记录下目前的时间，并根据实际情况填写附录 A：基于 PSP 2.1 修改的 PSP 表格的“实际耗时”栏目。

→  **Q2.7(I)** 请写下本部分的心得体会。

Chapter.3 这就是我的前进、到我出场了！！！！！！（It's MyGO!!!!!!）

 本章内容的参考完成时间为 360 分钟。

→  **Q3.1(P)** 请记录下目前的时间。

→  **Q3.2(P)** 请在完成任务的同时记录，并在完成任务后整理完善：

1. 浏览任务要求，参照附录 A：基于 PSP 2.1 修改的 PSP 表格，估计任务预计耗时；

2. 完成编程任务期间，依次做了什么（比如查阅了什么资料，随后如何进行了开发，遇到了什么问题，又通过什么方式解决）；

一同觅食一同战斗

（果子）仿佛观赏用的花朵一样 我只是远远地看着就满足了吗？

我还没有发现与你面对之后萌生的感情

越过障碍的蛇蛇们，在丛林里相遇了。越来越拥挤的山林深处，涌动着的是蛇蛇们的竞争心、还是蛇蛇们 **Amoris**（爱的）语言呢——

接下来将说明关于多蛇共斗的修订规则。

这些规则在  **T1** 的基础上增加/修改了以下内容，仅适用于本章任务。

注意不是在  **T2** 的基础上！本章任务中不会存在  **T2** 中定义的障碍物。

来吧，一起觅食吧？（新增/修改的实体定义）

- 场地
 - 场地的大小将发生变化。因为蛇蛇要战斗！场地会根据蛇的数量发生尺寸上的变化。本章中，场地上可能同时出现两条蛇或四条蛇。场地上出现两条蛇时，场地大小为 5×5 ；场地上出现四条蛇时，场地大小为 8×8 。
- 蛇
 - 撞别蛇的身子也会死！你将和其他蛇在同一个棋盘上，其他蛇的身子和头也会成为你的障碍物。你的蛇头与这些障碍物相撞（即重合）后，snake out。
 - 蛇头相撞同归于尽。根据蛇蛇第三定律，头的作用是相互的，所以两个蛇头相撞，两条蛇都 out。
- 果子
 - 果子的数量会根据蛇的数量发生变化。本章中，场地上可能同时出现两条蛇或四条蛇。场地上出现两条蛇时，果子数量为 5；场地上出现四条蛇时，果子数量为 10。
 - 果子的数量是守恒的。即每回合果子的数量不会减少。换言之，如果一个回合内果子减少了，下一回合开始时果子会生成并补充回一定数量。

来吧，一起战斗吧？（新的时序规则）

由于多蛇共斗的情况比较不一样，我们来从头完整地梳理一下规则吧！

回合制游戏

即便是多蛇共斗，依然是一款回合制游戏。你和其他蛇将会在一个回合内一起做出行动的决策，根据这些决策同时更新棋盘上的状态。

游戏开始

1. 游戏开始时，蛇被添加到棋盘上，标准测试能保证蛇出现的位置对称。两条蛇时，蛇分布在对角；四条蛇时，蛇分布在四个角上。
2. 随后，定量果子会随机出现在棋盘上，分布概率将会是一个均等分布。果子补充时概率与这里保持一致。

游戏进行

1. 决策：所有蛇根据此时场上的状况同时各自决策，决定蛇的移动方向。蛇的移动方向可能为上（+y方向）、左（-x方向）、下（-y方向）、右（+x方向）。给蛇一个指令吧！
2. 移动：所有蛇会按照自己的决策方向移动一格。蛇的头（第一节身体）移动到某个相邻的格子上，同时蛇头后的每节身体移动到上一刻前一节身体的位置。
3. 判定：判定此时每个蛇头的位置，蛇可能发生了碰撞或吃到了果子：
 - a. 碰撞：蛇会撞死在蛇的身体或场地边缘上。务请注意蛇的身体当然也包括其他蛇的身体。具体条件是：蛇的第一节身体（即蛇头）和蛇的任何一节身体重合，或者在场地以外（即与场地边界重合）。
 - i. 蛇撞死后，立即消失，将不会出现在下一回合的棋盘上。没有复活机会，所以生命很宝贵，千万不能早早离开啊。
 - ii. 注意当同时碰到障碍物和果子时，应该只有一种情况：即两蛇争食同一格。此时碰撞将优先判定，即两蛇都没了果子还在。
 - b. 得分：如果蛇吃到了果子，果子立即消失。本局游戏得分增加 1 分。
4. 补充：回合结束前，缺失的果子被补充。
 - a. 生成数量：果子数量是守恒的，将会在随机空格补充果子，场上差多少补多少。
 - b. 生成位置：生成位置互斥，即不会生成到当前时刻存在的果子和蛇上，保证不会超出棋盘边界。
5. 收尾：回合结束，除非满足了游戏结束条件，回到阶段 1。

游戏结束

满足下列游戏结束条件之一时，游戏结束。

1. 正常结局：达到回合数上限将会结束游戏，所有蛇的得分确定，开始结算决策时间。
2. 宇宙热寂：在回合数到达上限前，所有蛇都离开了我们。棋盘和果子都失去了意义——但是得分有意义，所以会开始计算所有蛇的得分和决策时间。

决策时间

大家的决策可以当作是行动时同步同时结算的，所以不会因决策时间先后对行动结果造成影响。

但是注意，决策时间将会设置上限 **T**，超过 **T** 时间内未决策将会有惩罚，而且决策时间将会影响最终评分。详细请看评价部分。

如果你还有疑惑的话，可以在课程群与助教确认规则。

蛇超绊

我寻着 寻着你我头碰头的果

再也不想松开手（蛇有手吗...？） 一直一直在一起吧

→ 🧑‍🔧 **T3 贪吃蛇 3：蛇蛇大作战！**

请根据以下要求及参考资料，完成任务。

本任务的全部源代码应该存放于 `/T3`。

代码有进展即进行 `commit`，`commit log` 不合理的项目会被助教抽查询问项目细节。

身经百战的我们的蛇蛇，决战群蛇之巅吧！


要求


你需要设计一个函数决定蛇的移动方式。

- 函数名: `greedy_snake_step()`、`greedySnakeStep()`，或参考你所用语言命名风格选择合适的命名。
- 参数: 为上述规则中所介绍的，“你看到的场地”的样子，由两个 `int32[]` 数组构成，依次为：
 - 棋盘大小 `n`
 - 一个 `i32` 类型数字，表示 $n \times n$ 的棋盘
 - 自己的蛇身坐标 `snake`
 - 每个坐标由两个 `i32` 构成，第一元为横坐标，第二元为纵坐标。数组的前两项为蛇头的坐标，之后依次是第二节身子、第三节身子、蛇尾（即第四节身子）的坐标。数组长度为 8。
 - 参数为 $(-1, -1) \times 4$ 时，代表蛇已经死亡，输出将被忽略。
 - 其他蛇的数量 `snake_num`
 - 一个 `i32` 类型数字
 - 一场游戏中，蛇的数量会在 $[0, \text{max_num}]$ 之间单调不增。
`max_num` 为游戏开始时的蛇数量。
 - 其他蛇的坐标
 - 一个长度为 $\text{snake_num} \times 8$ 的 `i32` 类型数组，表示 `snake_num` 条蛇的坐标。
 - 蛇的坐标定义同理。务请注意死蛇不会作为参数传入。
 - 果子的数量 `food_num`
 - 一个 `i32` 类型数字
 - 果子的坐标 `foods`
 - 一个长度为 $\text{food_num} \times 2$ 的 `i32` 类型数组，依次表示 `food_num` 个果子的坐标。每一对坐标对的第一元为横坐标，第二元为纵坐标。
 - 剩余回合数 `round`
 - 一个 `i32` 类型数字，表示包含本次决策在内的剩余回合数。
 - 本次决策结束后，`round` 将会减一，即传入 `round` 为 1 时，代表这是最后一回合。

- 返回值：移动方向 `direction`
 - 一个 `i32` 类型数字，代表蛇本轮选择的移动方向。
 - 输出值为 `0, 1, 2, 3` 四个数字中的一个，分别对应上（`y`）、左（`-x`）、下（`-y`）、右（`y`）四个方向。
- 其他要求：
 - 你的程序实现应该完全在本地运行，不能通过联网等方式使用远程计算资源。如果对程序实现的合法边界有任何疑问，请随时与课程组确认，课程组保留对合法决策手段的解释权。
 - 每一次调用该函数的时间（端到端计时），不得超过 **T=500 ms**。你可以参考后文的评测环境描述；
 - 如果你们的程序实现时间已经达到了 8 小时以上，请尽快收尾好好休息！本项目的设计不期望大家无谓地内卷性能分，投入更多对大家的课程体验和学习收获收益微乎其微。


→  **Q3.3(P)** 请说明你们如何建模这一需求。

→  **Q3.4(P)** 请说明针对该任务，你们采取了哪些策略来优化决策。具体而言，怎么避免死亡？怎么吃到更多果子？如何编程实现。

→  **Q3.5(P)** 请说明你们如何量度所实现的程序模块的有效性，例如：“如何说明我们的程序模块对局能力很强？”尝试提出一些可能的定量分析方式。

测试

当你完成代码编写任务后，请切换到 `/T3` 目录下：

1. 联系其他已完成  T3 的小组，请他们提供编译出的 `Wasm` 模块及 `JS` 胶水代码（不允许互相交换源代码！），将相关文件粘贴到新建文件夹 `t3-snake-{num}-{PL}`。如果实在联系不上其他小组，请将自己完成的代码，复制几份存放在新建文件夹 `t3-snake-{num}-{PL}` 下；
2. 查看并根据需要修改 `game-config.js` 文件，该文件包含了以下测试配置：
 - 游戏模式选择 (`GAME_MODE`) - `"1v1"`, `"4snakes"` 或 `"custom"`
 - 蛇的决策函数导入和映射 (`snakeModules`)，请尝试导入你的模块和其他小组的模块
 - 不同模式下的游戏参数配置 (`gameParameters`)，包括棋盘大小、蛇数量、果子数量、最大回合数及初始蛇位置

- 随机种子设置 (`CUSTOM_SEED`) - 可以设置为特定值或 `undefined` 表示随机生成, 指定 `seed` 可以用于确定化复现游戏结果。
- 蛇的显示配置 (`SNAKE_DISPLAY_CONFIG`) - 符号和颜色

3. 然后, 执行以下命令来完成提交前的测试:

```
npm run submit-test
```

如果测试输出如下, 说明你们的代码实现已被接受作为有效提交。

```
=== FINAL RESULTS ===
Snake scores:
Snake 1: {得分} points {(survived) 或 (died in round N)} spent {决策总时间}ms
Snake 2: {得分} points {(survived) 或 (died in round N)} spent {决策总时间}ms
// ...
```

测试可视化

你还可以使用TUI界面查看游戏进程:

```
# 需要安装一次依赖
npm install
npm run display-test
```

标准测试

我们在配置文件中提供了两种标准测试, 你可以直接通过修改 `GAME_MODE` 来使用, 后续的评分也只会使用这两种标准测试:

- **1v1**: 蛇的数量为 2, 棋盘大小为 5, 果子数量为 5, 回合数为 50。
- **4 蛇大乱斗模式**: 蛇的数量为 4, 棋盘大小为 8, 果子数量为 10, 回合数为 100。

初始蛇的位置已在配置文件中设置。

自定义测试

如果你想测试更多种配置的情况，可以设置 `GAME_MODE` 为 "custom"，并根据需要调整 `gameParameters` 的 "custom" 部分参数。

评价

满分 40 分，课程组将会分为两个阶段进行评价：


- **1v1 模式：**将对全部结对小组的程序进行两两对战，记录对战积分 n ，以及决策总时间 t 。
 - 考虑随机种子的影响，每 2 组会进行随机种子不同的 4 次对战，计算 4 局两条蛇的总得分（进食的果子数加和）。总得分较多者对战积分增加一分；
 - 一对对手 4 次对战总得分相同，则按照总决策时间决定哪条蛇获得对战积分。
 - 以对战积分 n 排名，胜出得分 n 相同的队伍，将会按照决策总时间 t 进行排序。
 - 前 10 名将晋级到 4 蛇大乱斗模式；
 - 末 5 名获得 5 分得分；
 - 其他队伍依次获得 11-30 分的评分。
 - 如果边界出现同分情况，但决策时间差距小于 3%：对于晋级边界的队伍将有机会晋级第二轮，最多捞入边界前两队；对于末分边界的队伍将获得 10 分，应捞尽捞。
- **4 蛇大乱斗模式：**将会对晋级的 10 支队伍进行 C_{10}^4 次全遍历的对战，记录对战积分 n 及决策总时间 t 。
 - 考虑位置分布的影响，每 4 队将会变更位置进行 6 次对战（想想圆排列！），每次对战将会按照局内得分（进食的果子数）进行排名，结算对战积分，第 1/2/3/4 名分别获得 3/2/1/0 对战积分。注意与两两对战不同，不是 6 次对战加和结算，而是 6 次对战分别结算，所以是结算 6 次分。
 - 局内得分相同，则按照本局决策时间进行排序；决策时间差距小于 3%，平分两个名次的分数。
 - 以对战积分 n 排名，对战积分 n 相同的队伍，将会按照决策总时间 t 进行排序。
 - 最终 10 支队伍将获得 33~42 分的评分（即可能溢出！最多能有 3 支队伍有望获得结对项目的满分）。


- 超时或异常情况：
 - 时间上限 T 为 **500ms**。
 - 若你的程序超时做出决策，将会按照当前的方向继续移动（即蛇的第一节和第二节所指向的方向），且按照实际决策时间计入总时间。
 - 若你的程序陷入死循环（单步超时 2000 ms），或是崩溃，无法得出结果。
 - 此次对战会将你判定为最后一名，决策时间按照 总回合数 $\times T$ 计算。
 - 刨去你的蛇之后，其他蛇重新开一盘缺位的4蛇大乱斗，可以视作你的蛇在第一局脖子后拧死亡。

标准评测环境

将采取并行评测，请适当预留评测余量。评测环境根据预期情况可能发生一些改变，但不会发生太大的变化。

- **HARDWARE SPEC:** CPU: AMD EPYC 7763 [$\times 2$], 64 [$\times 2$] cores. With 1T RAM, on SSD. **GPU will be disabled.**
- **SOFTWARE SPEC: OS:** Ubuntu 20.04; **JS Runtime:** Node.js LTS (now at v22.14.0).

→  **Q3.6(P)** 请记录下目前的时间，并根据实际情况填写附录 A：基于 PSP 2.1 修改的 PSP 表格的“实际耗时”栏目。

→  **Q3.7(I)** 请写下本部分的心得体会。

其他评价方式

晋级第二轮的每个小组将获得长得有点像蛇蛇的，珍宝珠彩虹酸条软糖 $\times 2$ 。


结语

一起吃一辈子果子吧——！

→  **Q4.1(P)** 提供两人在讨论的结对图像资料。

→  **Q4.2(P)** 回顾结对的过程，反思有哪些可以提升和改进的地方。

→  **Q4.3(I)** 锐评一下你的搭档！并请至少列出三个优点和一个缺点。

→  **Q4.4(I)** 说明结对编程的优缺点、你对结对编程的理解。

→  **Q4.5(P)** 请提供你们完成代码实现的代码仓库链接。

附录

附录 A：基于 PSP 2.1 修改的 PSP 表格

PERSONAL SOFTWARE PROCESS STAGES	个人软件开发流程	预估耗时 (分钟)	实际耗时 (分钟)
PLANNING	计划		
- Estimate	- 估计这个任务需要多少时间		
DEVELOPMENT	开发		
- Analysis & Design Spec	- 需求分析 & 生成设计规格（确定要实现什么）		
- Technical Background	- 了解技术背景（包括学习新技术）		
- Coding Standard	- 代码规范		
- Design	- 具体设计（确定怎么实现）		
- Coding	- 具体编码		
- Code Review	- 代码复审		
- Test Design	- 测试设计（确定怎么测，比如要测试哪些情景、设计哪些种类的测试用例）		
- Test Implement	- 测试实现（设计/生成具体的测试用例、编码实现测试）		
REPORTING	报告		
- Quality Report	- 质量报告（评估设计、实现、测试的有效性）		
- Size Measurement	- 计算工作量		
- Postmortem & Process Improvement Plan	- 事后总结和过程改进计划（总结过程中的问题和改进点）		
TOTAL	合计		

声明和致谢

本项目由 2025 级北京航空航天大学计算机学院敏捷软工课程组设计。

项目设计：@Kkkk_qy, @TobyShi, @volcaXiao, @YtZhong, @KumaXX；标程：@Kkkk_qy, @TobyShi, @volcaXiao, @YtZhong, @BlueBean；审稿校稿润色：@KumaXX；评测设计和支持：@Kkkk_qy, @TobyShi, @volcaXiao, @YtZhong, @KumaXX；idea 提供（万恶之源）：@coder014

感谢 @Red, @BlueBean, @Chenrt 和其他关心项目的大家对项目的建议！