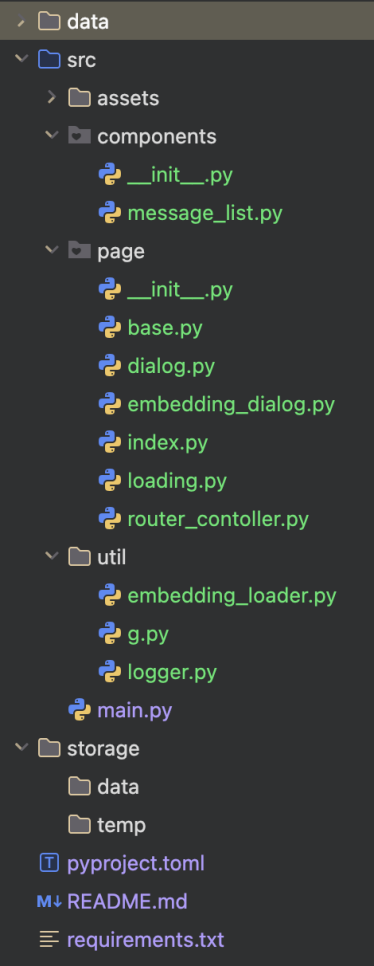
**个人工作报告**

一、工作内容概述

在本次AI飞花令系统开发项目中，我作为项目组员，主要使用可视化界面开发、各方案集成调用，以及回报展示。具体工作如下：

1. 可视化界面开发：使用Flet库进行可视化界面的开发。Flet在Flutter中集成了python运行时，使得python可以编写Flutter界面，为用户提供可视化的“飞花令”游戏的体验。
2. 方案集成调用：使用python abc模块中的abstractmethod实现抽象方法，构造抽象类，提供后续调用的接口，为GTP、词向量、爬虫等多种方案集成提供同一调用接口。
3. 工作完成过程
4. 项目的架构设计



1. page文件夹：各个页面的类，继承Flet.View
2. components文件夹： 各个组件，MessageList类继承ListView，用于渲染对话界面。
3. util文件夹：各个工具类，embedding\_loader提供词向量的加载，g是全局变量，用于保存加载后的模型，logger封装日志，采用单例模式。

本项目严格使用type\_hint，减少因为python的动态类型带来的开发效率问题。

1. Flet可视化界面开发

Flet模块拥有同步、异步两种开发方案，但是Flet未提供初始化的接口，异步开发不得不使用工厂函数，会极大的增加开发量，因此本项目采用同步方案。

1. 主函数

flet需要传入一个主函数，函数要求page: Flet.Page类型的参数。本项目主函数主要是用于初始化RouterController，提供路由管理。

1. 路由管理器

class RouterController:  
 PAGES: List[Type[BasePage]] = [  
 IndexPage,  
 LoadingPage,  
 EmbeddingDialogPage  
 ]  
  
 def \_\_init\_\_(*self*, *page*: ft.Page):  
 *self*.page = *page  
 self*.page.on\_route\_change = *self*.on\_route\_change  
 *self*.page.go(IndexPage.ROUTE)  
  
 def on\_route\_change(*self*, *route*: RouteChangeEvent):  
 *self*.page.views.clear()  
 route = urllib.parse.urlparse(*route*.route)  
 for page in *self*.PAGES:  
 if *route*.path == page.ROUTE:  
 *self*.page.views.append(  
 page(  
 page=*self*.page,  
 \*\*{key: value for key, value in urllib.parse.parse\_qsl(*route*.query)}  
 )  
 )  
 *self*.page.update()

路由管理器提供动态的路由访问，当页面路由发生改变时调用on\_route\_change函数。所有的页面都集成自BasePage，拥有静态属性ROUTE、继承自Flet.View。当路由变化时，通过urlib模块解析匹配路由并通过解包来传递参数（Flet未提供动态路由的标准接口）

1. 对话基类

class DialogPage(BasePage):  
 ROUTE: *str* = "/dialog"  
  
 def \_\_init\_\_(*self*, *page*: ft.Page, *title*,  
 *…*  
  
 def on\_send(*self*, *\_*, *text*: *str*) -> List[*str*]:  
 …  
  
 @abstractmethod  
 def get\_text(*self*, *index*: *int*) -> Optional[*str*]:  
 raise *NotImplementedError* @staticmethod  
 def on\_back\_index(*event*: ft.ControlEvent):  
 *event*.page.go("/")  
  
 def on\_click(*self*, *evnet*: ft.ControlEvent):  
 …

对话基类提供了get\_text的接口，用于子类的扩展，方便项目后续的二次开发和更新，同时规范接口，当返回为None时，接口无数据返回，判断游戏结束。

例如，下面是词向量方案的实现：

class EmbeddingDialogPage(DialogPage):  
 ROUTE: *str* = "/embedding"  
  
 def \_\_init\_\_(*self*, *page*: ft.Page, *title*: *str*, *value*: *str*):  
 *self*.result = GlobalData.EmbeddingLoader.search\_poems(*value*)  
 *super*().\_\_init\_\_(*page*, *title*, *value*)  
  
 def get\_text(*self*, *index*: *int*) -> Optional[*str*]:  
 for i in *self*.result:  
 if *len*(i) == *self*.len\_of\_line and *len*(i) > *index* and i[*index*] == *self*.value:  
 return i  
 return None

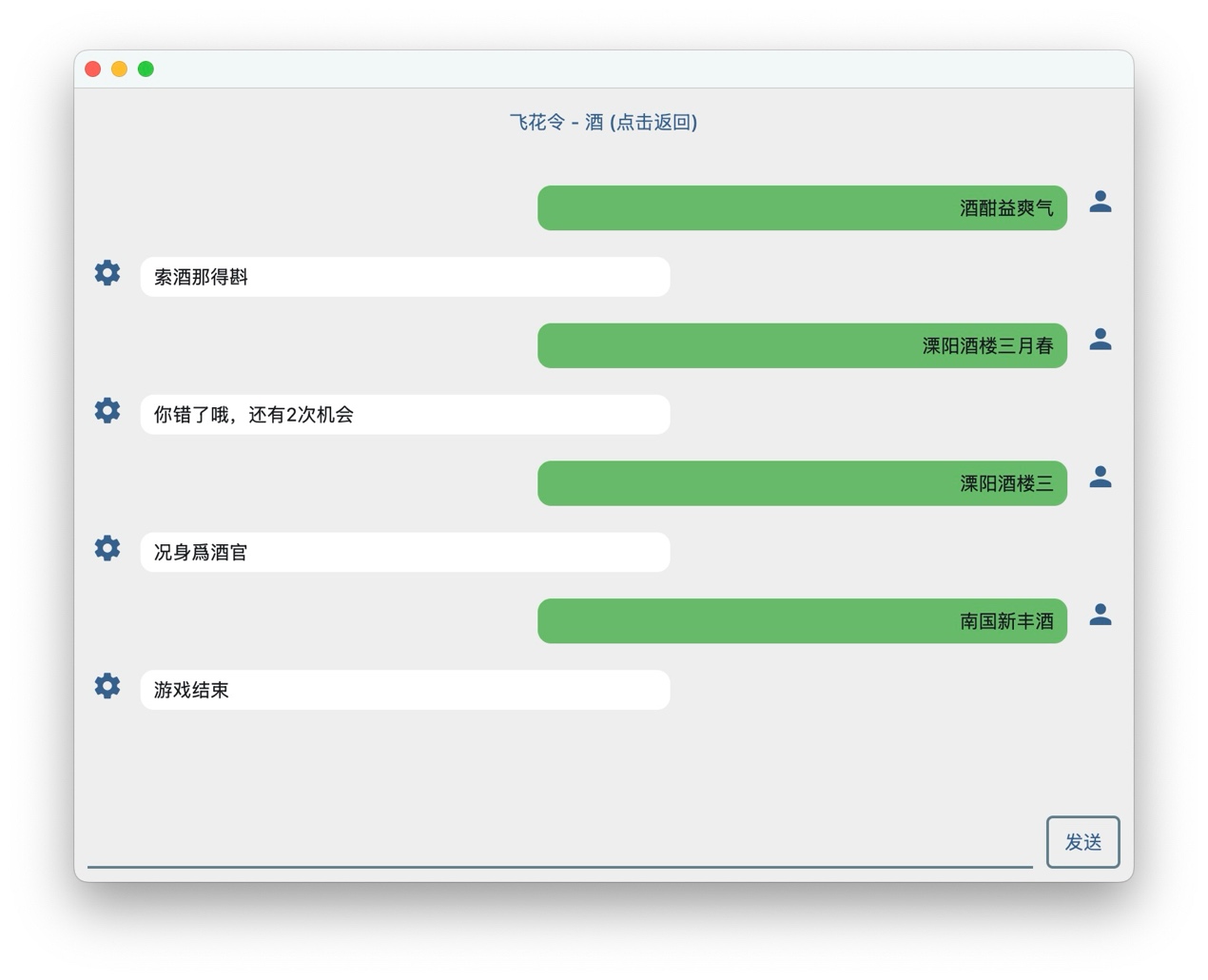
1. 配置加载器

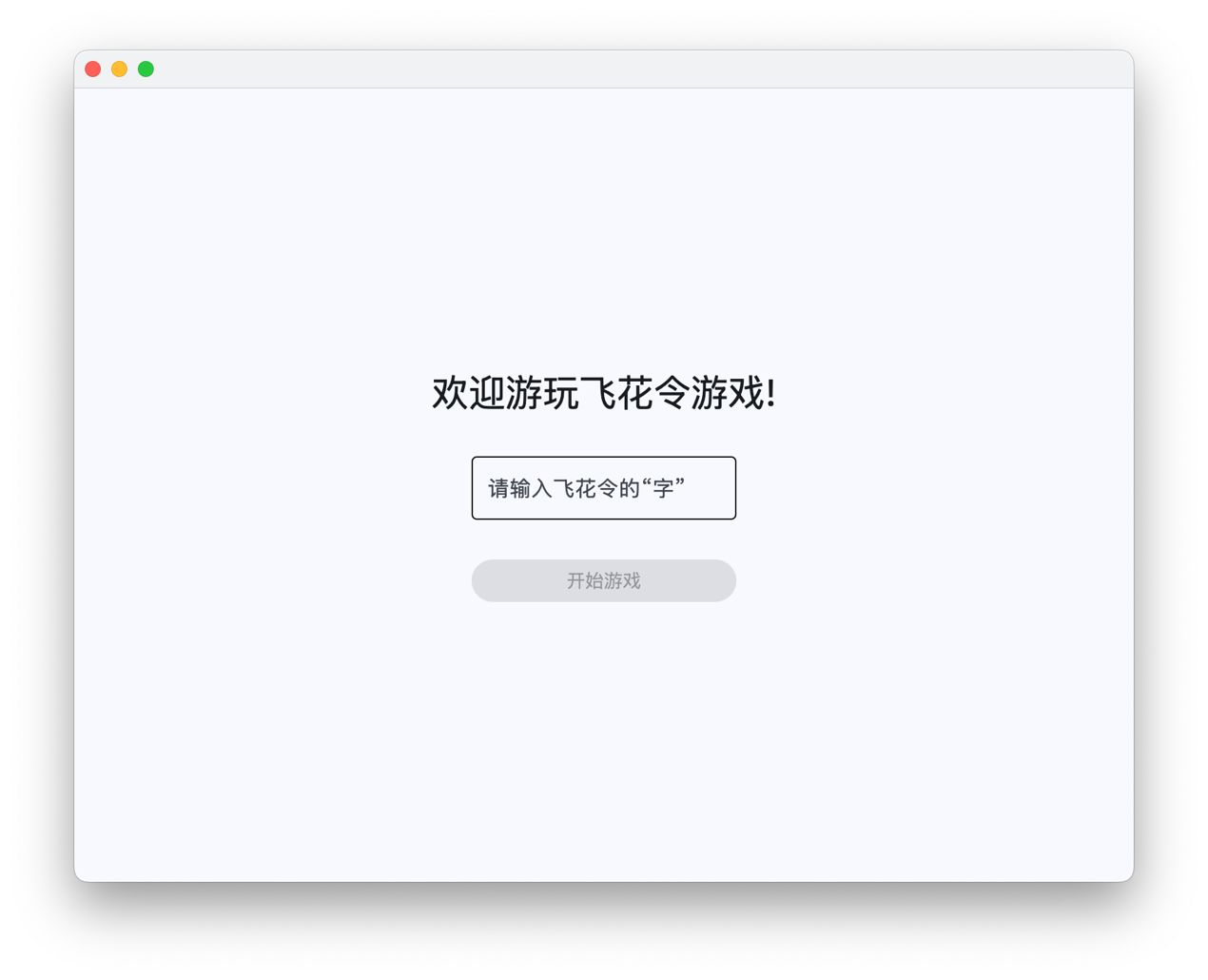
配置加载器用于检查配置的状态，并在日志中输出对应的错误。使用pydantic对配置据进行验证，同时编写逻辑代码对配置进行验证。

class EmbeddingLoader:  
 *"""  
 用于加载词向量模型  
 """* def \_\_init\_\_(*self*, *config*: EmbeddingConfig):  
 if (os.path.exists(path=*config*.poem\_sentences\_path) is False or  
 os.path.isfile(path=*config*.poem\_sentences\_path) is False):  
 Logger.new().error(f"加载poem\_sentences失败，文件不存在: {*config*.poem\_sentences\_path}")  
 raise *FileNotFoundError*(f"{*config*.poem\_sentences\_path}")  
 with *open*(*config*.poem\_sentences\_path, "rb") as f:  
 *self*.poem\_lines = pickle.load(f)  
 Logger.new().info(f"加载poem\_sentences成功")  
  
 if (os.path.exists(path=*config*.poem\_embeddings\_path) is False or  
 os.path.isfile(path=*config*.poem\_embeddings\_path) is False):  
 Logger.new().error(f"加载poem\_embeddings失败，文件不存在: {*config*.poem\_embeddings\_path}")  
 raise *FileNotFoundError*(f"{*config*.poem\_embeddings\_path}")  
 embeddings = np.load(*config*.poem\_embeddings\_path)  
  
 *self*.model = SentenceTransformer("shibing624/text2vec-base-chinese", cache\_folder=*config*.cache\_path)  
  
 *self*.index = faiss.IndexFlatL2(embeddings.shape[1])  
 *# noinspection PyArgumentList  
 self*.index.add(embeddings)  
 Logger.new().info(f"加载poem\_embeddings成功")

三、工作成果与收获总结

基本实现视图展示，经过尝试Flet库没有绑定BoxConstrains，无法实现对话框的长度动态化。实现接口的规范化，为诗句来源提供接口。经过测试，“飞花令”游戏可以进行游玩，如下图所示：





项目同时存在着一些问题。在加载的过程中存在一些耗时操作，由于跨线程进行操作，可能会导致数据丢失问题，同时在县城结束后会有反复刷新造成的视图卡顿操作，使用异步开发可以减少这部分的问题。

四、未来展望

本次项目让我意识到项目架构的重要性，在python中使用type\_hint、装饰器可以有效提升开发效率。在以后的项目中尽量使用异步的接口，在处理耗时操作（如模型加载）有更好的效果，减少线程竞争带来的全局变量、局部变量不一致问题。