数据库实验报告

黄绵秋 19307130142 宋宇霖 19307130139

一、实验题目

宝可梦小游戏的设计与实现

有谁会不喜欢可可爱爱的精灵宝可梦呢？为此我们设计了一个简易版的宝可梦小游戏。游戏中管理员可以通过宝可梦世界的发展来调整宝可梦数据和道具的数据，加入或者删除宝可梦，用户可以通过对战捕捉宝可梦，收藏自己喜欢的宝可梦！

二、开发环境

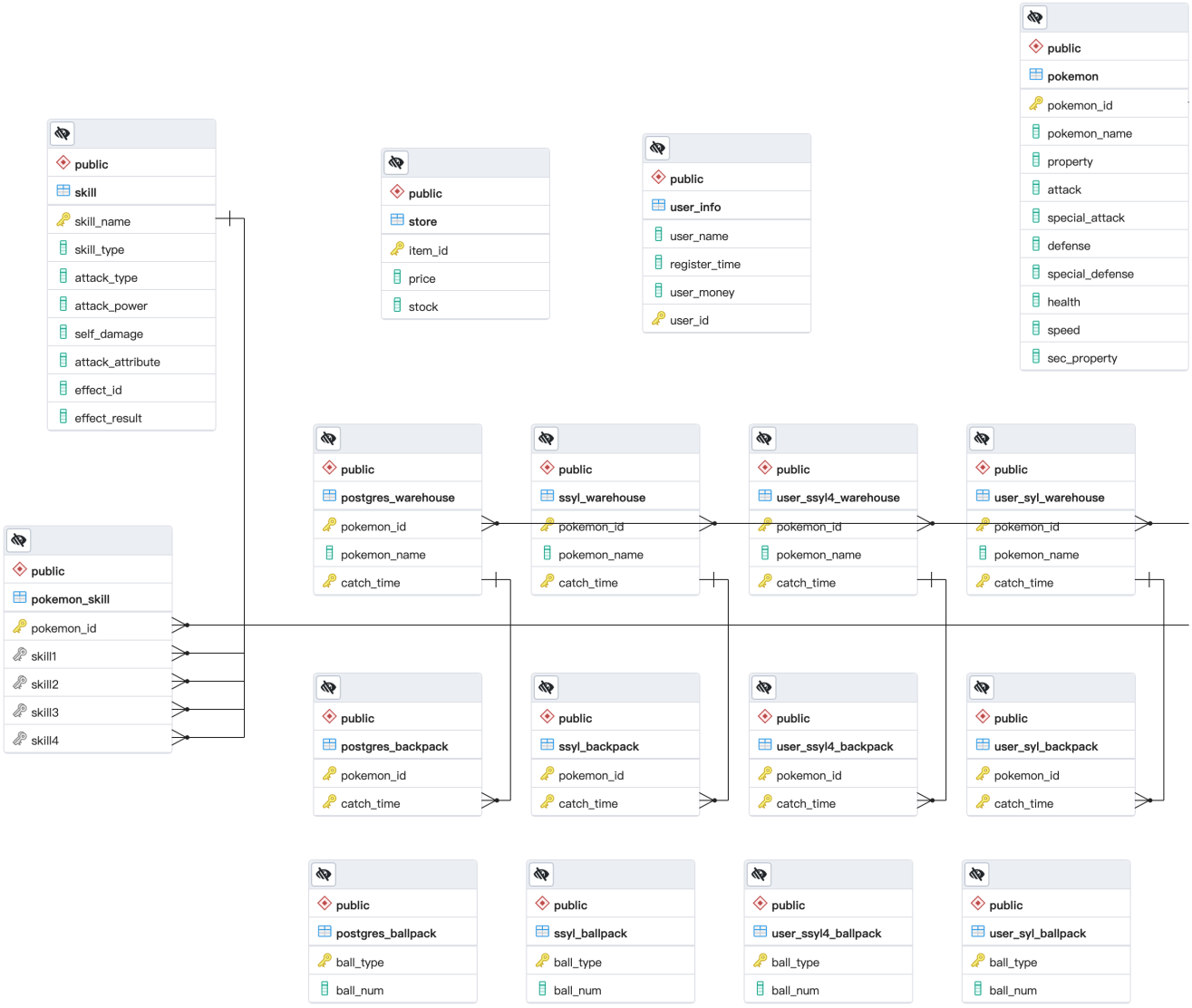
介绍本次实验你所采用的开发环境，包括操作系统，数据库管理软件，编程语言，客户 端或 Web 开发环境等。

本次实验于mac os系统上开发，数据库管理软件为postgresql，后端的编程语言为golang，web开发基于http协议，使用了gin框架方便开发。前端采用CocosCreater游戏引擎

三、数据库设计

介绍本项目的数据库设计，给出系统数据库的ER 图，对每个表的功能和属性进行说明。

ER图



注：原始ER图由pgadmin导出，根据课本ER图的画法手动加了一些关系。🔑图标为主码。由于我们采用的是以用户名为基础建立用户自己的表的方式，因此在数据库中，用户信息表中的元组和对应仓库背包没有直接的关系，但是和数据库的用户是有关系的，在go语言的处理逻辑中也是有关系的。

表介绍

1. **pokemon**

这个表存储了所有宝可梦的信息和属性：包括主码pokemon_id，pokemon_name, property, sec_property(不是每个宝可梦都有)；以及各项能力，包括attack, special_attack，denfense, special_defense, health, speed。

功能：作为游戏的宝可梦指南，用于用户的查询和了解宝可梦信息。

2. **pokemon_skill**

这个表存储了宝可梦id和技能，包括主码pokemon_id,几个技能:skill1, skill2, skill3, skill4

功能：在对战的时候通过宝可梦id找到宝可梦拥有的技能并显示，同时也供用户了解 自己的宝可梦的技能

3. **skill**

这个表存储了每个技能的信息，伤害和影响。包括skill_name（主码），skill_type（决定是伤害型技能还是攻击型技能），attack_type（攻击类型）, attack_power（攻击威力），self_damage（自我伤害）, attack_attribute（攻击属性）, effect_id（产生的影响的id），effect_result（产生的影响的效果），后两个属性传回前端由前端的对战函数进行处理，具体效果就是根据不同效果对于对战精灵的属性值进行更改。我们在设计之初，effect_id和effect_result设计的是多值属性，类型为[]int，这是因为一个技能可以有多重不同的效果。但是在尝试传参时，scan函数并不能把多值属性传回后端的go语言中以数组形式存放。我们尝试了多种方法均没有成功，因此最终不得不改成单值属性，简化了这部分的逻辑。

功能：存储所有功能的基本信息，这些信息会传到前端，由前端的对战逻辑处理函数决定对于对战的情况产生什么影响

4. **store**

这个表存储商店的商品。为了尊重宝可梦原著的思想，我们不提供宝可梦购买服务，只提供精灵球。精灵球分为四种，以主码item_id作为区分，另有属性price和stock分别表示价格和库存。

5. **user_info**

这个表存储玩家的基本信息，包括主码user_name，和注册时间register_time，用户的金钱user_money

6. **postgres_warehouse**

这个表是用户postgres的宝可梦仓库，存放着用户拥有的宝可梦的信息，包括（pokemon_id, catch_time)（主码）和pokemon_name

功能：用户可以从这个仓库选择宝可梦放入自己的背包，并可以在仓库里查看自己的宝可梦的信息，提升用户的参与感和获得感。

7. **postgres_backpack**

这个表是用户的宝可梦背包，里面存放着用户的宝可梦。属性有主码（pokemon_id, catch_time）用于表明精灵的身份。加入catch_time是因为用户可以拥有多只相同的精灵，为了区分它们，就在主码里加上了抓到的时间。

功能是带着宝可梦进行对战。

8. **postgres_ballpack**

这个表是用户的精灵球背包，里面存放着四种精灵球，以及拥有的数量。属性有主码ball_type和另一个属性ball_num

功能是带着精灵球用于捕捉。

注：**6,7,8三个表的前缀用户名都只是示意**，对于每一个注册的用户user，都会有自己的user_backpack表，由于属性和功能相同，故不加赘述。**这三个表在都在注册用户的时候直接创建，并且将所有者设定为改用户。**

四、系统设计

对照实验要求，展示系统的主要功能，适当截图，对每个功能内部的实现流程进行说明。 对某些关键的功能实现，可以通过展示代码来分析。

以下结合代码进行展示，功能较简单或者类似的代码，为了避免报告冗长就不展示了

具体的图形界面和游戏机制将在现场展示，就不截图了

以下是后端中的函数展示。部分功能并不体现在前端界面上（因为是管理员功能或者加载用的功能，我们制作的是游戏的客户端，因此暂时不提供管理员图形界面，但是可以通过命令窗来实现对游戏进行管理）

此外，前端中对于信息的接收和处理有大量代码，但鉴于包括对战在内的不少内容均与数据库交互无关，所以报告中不予展示，但会在演示时进行介绍。

```
//后段部分
router.POST("/select", Select_handler)//管理员和用户查询宝可梦信息
router.POST("/insert", Insert_handler)//管理员加入新的宝可梦
router.POST("/delete", Delete_handler)//管理员删除宝可梦
router.POST("/update", Update_handler)//管理员更新宝可梦信息
router.POST("/create_role", Create_role_handler)//管理员创建账户
router.POST("/login", Login_handler)//用户登录
router.POST("/user_free", Poke_free_handler)//用户放生宝可梦
router.POST("/takein", Takein_handler)//用户从仓库里将宝可梦加入背包
router.GET("/warehouse", Warehouse_handler)//用户查看仓库信息
router.POST("/takeout", Takeout_handler)//用户从背包里取出宝可梦
router.POST("/buy", Buy_pokeball_handler)//用户购买精灵球
router.POST("/catch", Catch_handler)//用户抓到了宝可梦，加入仓库
router.POST("/award", Award_handler)//记录和处理用户得到的奖励
```

```
router.POST("/fight", Fight_handler)//加载用户发生战斗时的，以及AI宝可梦的信息
router.POST("/register",Register_handler)//用户注册
router.GET("/user_info", User_info_handler)//提供用户的基本信息
router.GET("/logout",Logout_handler)//用户退出登录
//router.GET("/delete_user", Delete_user_handler)//删除用户
```

```
//前端部分
module http //该模块负责向后端发送请求
module Constant //该模块负责存储与后端交互时所用的url
//调用方式如下

var obj = {
    'url': Constant.LOGIN_URL,  // 指明请求的url
    'method': 'POST',      // 指明通信为GET或POST
    'data': {          // 指明传输的数据，采用json进行数据传输
        'id': this.ID,
        'password': this.PassWord
    },               // 返回状态码为200（交互成功）时执行的函数
    'success': function(jsonData) {
        this._onLoginSuccess(jsonData)
    }.bind(this),

    'fail': function(jsonData) {// 交互失败时执行的函数
        this._onLoginFail(jsonData)
    }.bind(this)
}
http.request(obj)        // 发送请求
```

接下来对功能进行详解

管理员功能

- 查看宝可梦信息

```
func Select_handler(c *gin.Context) {
  var info Pokemon_info
  c.ShouldBindJSON(&info)
  fmt.Println(info)
```

```
  if info.Id == "" {
    info.Id = "0"
  }
  if info.Name == "" {
    info.Name = "0"
  }
  if info.Property1 == "" {
    info.Property1 = "0"
  }
  if info.Property2 == "" {
    info.Property2 = "0"
  }
  //由于GO语言的struct无法赋初值，只好在这里设置初值
  infos, i := pokemon_select(info) //调用函数获取select得到的信息，该函数代码下面
会介绍
  var my_slice []Pokemon_info
  for j := 0; j < i; j++ {
    my_slice = append(my_slice, infos[j])
  }
  c.IndentedJSON(200, my_slice) //以json数组格式返还给前端
}

//根据生成的查询语句，查询并且读出信息写入infos结构数组返回
func pokemon_select(info Pokemon_info) ([1000]Pokemon_info, int) {
  sentense := select_make(info)//根据传入的信息生成查询语句
  rows, err := db.Query(sentense)
  checkErr(err)
  var infos [1000]Pokemon_info
  i := 0
  for rows.Next() {
    err = rows.Scan(&infos[i].Id, &infos[i].Name, &infos[i].Property1,
      &infos[i].Attack, &infos[i].Special_Attack, &infos[i].Defense,
&infos[i].Special_Defense,
      &infos[i].Speed, &infos[i].Health, &infos[i].Property2)
    i += 1
  }
  return infos, i
}

//生成查询语句
func select_make(info Pokemon_info) string {
  str := "select* from pokemon "
  tag := 0 //用来判断语句是否要加and
```

```
  if info.Id != "0" {
    if tag != 0 {
      str += " and"
    } else {
      str+="where"
      tag = 1
    }
    str += " pokemon_id= '" + info.Id + "'"
  }
  if info.Name != "0" {
    if tag != 0 {
      str += " and"
    } else {
      str+="where"
      tag = 1
    }
    str += " pokemon_name= '" + info.Name + "'"
  }
  if info.Property1 != "0" {
    if tag != 0 {
      str += " and"
    } else {
      str+="where"
      tag = 1
    }
    str += " property= '" + info.Property1 + "'"
  }
  if info.Property2 != "0" {
    if tag != 0 {
      str += " and"
    } else {
      str+="where"
      tag = 1
    }
    str += " sec_property= '" + info.Property2 + "'"
  }
  if info.Attack != 0 {
    if tag != 0 {
      str += " and"
    } else {
      str+="where"
      tag = 1
    }
```

```
    str += "\nattack=" + strconv.Itoa(info.Attack)
  }
  if info.Special_Attack != 0 {
    if tag != 0 {
      str += " and"
    } else {
      str+="where"
      tag = 1
    }
    str += " special_attack=" + strconv.Itoa(info.Special_Attack)
  }
  if info.Defense != 0 {
    if tag != 0 {
      str += " and"
    } else {
      str+="where"
      tag = 1
    }
    str += " defense=" + strconv.Itoa(info.Defense)
  }
  if info.Special_Defense != 0 {
    if tag != 0 {
      str += " and"
    } else {
      str+="where"
      tag = 1
    }
    str += " special_defense=" + strconv.Itoa(info.Special_Defense)
  }
  if info.Health != 0 {
    if tag != 0 {
      str += " and"
    } else {
      str+="where"
      tag = 1
    }
    str += " health=" + strconv.Itoa(info.Health)
  }
  if info.Speed != 0 {
    if tag != 0 {
      str += " and"
    } else {
      str+="where"
```

```
        tag = 1
      }
      str += " speed=" + strconv.Itoa(info.Speed)
    }
    return str
}
```

- 插入宝可梦： 功能是对于宝可梦图鉴（表pokemon）进行插入，加入新的宝可梦。
- 删除宝可梦：功能是对于宝可梦图鉴进行删除，删去原有的宝可梦
- 更新宝可梦：功能是对于宝可梦图鉴进行修改，更改宝可梦信息。会判断传入的宝可梦主码是不是新的，如果是就更新，不是就插入一个新的宝可梦
- 以上三个功能代码和查询类似，所以就不贴上代码了，具体见源码
- 新建用户，功能是新建一个用户，并且进行初始化操作，具体有：

```
func Create_role_handler(c *gin.Context) {
  var user User_info
  c.ShouldBindJSON(&user)
  user.Id = "user_" + user.Id
  user.Password = encode_password(user.Password)//对密码加密
  err2 := create_role(user)
  if err2 == nil {
    c.String(http.StatusOK, "create a role successfully")
  } else {
    c.String(http.StatusNotAcceptable, "fail to create a role")
  }
}

//以上函数调用了以下函数：
func create_role(user User_info) error {
  str := "CREATE ROLE \"" + user.Id + "\" with\n"
  if user.Login != 0 {
    str += "login\n"
  } else {
    str += "nologin\n"
  }
  if user.Superuser != 0 {
    str += "superuser\n"
  } else {
```

```
    str += "nosuperuser\n"
  }
  if user.Createdb != 0 {
    str += "createdb\n"
  } else {
    str += "nocreatedb\n"
  }
  if user.Createrole != 0 {
    str += "createrole\n"
  } else {
    str += "nocreaterole\n"
  }
  if user.Inherit != 0 {
    str += "inherit\n"
  } else {
    str += "noinherit\n"
  }
  if user.Replication != 0 {
    str += "replication\n"
  } else {
    str += "noreplication\n"
  }
  str += "connection limit -1\n"
  str += "password '" + user.Password + "';"
  _, err := db.Query(str)
  checkErr(err)
  //以上部分生成了用户的创建语句，根据传回的参数选择用户的权限
  if err == nil {
    //在创建用户的时候为用户新建一个仓库
    sentence := "create table " + user.Id + "_warehouse (\n"
    sentence += "pokemon_id character varying(20),\n"
    sentence += "pokemon_name character varying(20),\n"
    sentence += "catch_time timestamp,\n"
    sentence += "primary key(pokemon_id,catch_time),\n"
    sentence += "foreign key(pokemon_id) references
pokemon(pokemon_id)\n );"
    _, err = db.Query(sentence)
    //创建用户的backpack
    sentence3 := "create table " + user.Id + "_backpack (\n"
    sentence3 += "pokemon_id varchar(20),\n catch_time timestamp,\n " +
      "primary key(pokemon_id,catch_time)," +
      "\n\tforeign key(pokemon_id,catch_time) references " + user.Id +
"_warehouse(pokemon_id,catch_time));"
```

```
    _, err = db.Query(sentence3)
    //创建用户的ballpack
    sentence4 := "create table " + user.Id + "_ballpack (\n"
    sentence4 += "ball_type int,\n\tball_num int,\n\tprimary
key(ball_type));"
    _, err = db.Query(sentence4)
    //在ballpack里加上四种精灵球的元组
    sentence5 := "insert into " + user.Id + "_ballpack values(1, 0);\n"
    sentence5 += "insert into " + user.Id + "_ballpack values(2, 0);\n"
    sentence5 += "insert into " + user.Id + "_ballpack values(3, 0);\n"
    sentence5 += "insert into " + user.Id + "_ballpack values(4, 0);"
    _, err = db.Query(sentence5)
    //在用户信息中加入新用户
    sentence1 := "insert into User_info values('" + user.Name + "',
now(), 0, '" + user.Id + "');\n"
    _, err = db.Query(sentence1)
    //设置两个背包和仓库表的所有者为用户，从而获得权限
    sentence2 := "ALTER TABLE " + user.Id + "_warehouse\n OWNER to " +
user.Id + ";\n"
    sentence2 += "alter table " + user.Id + "_backpack\n owner to " +
user.Id + ";\n"
    sentence2 += "alter table " + user.Id + "_ballpack\n owner to " +
user.Id + ";"
    _, err = db.Query(sentence2)
    //设定用户对于五个主表的权限，其中对于个人信息可以查看和改，对于其他表只能查看。
商店表设置为可以更新是为了方便商店库存的管理
    sentence6 :="GRANT SELECT ON TABLE public.pokemon TO " + user.Id +
";\n"
    sentence6 += "GRANT SELECT ON TABLE public.pokemon_skill TO " +
user.Id + ";\n"
    sentence6 += "GRANT SELECT ON TABLE public.skill TO " + user.Id +
";\n"
    sentence6 += "GRANT SELECT,UPDATE ON TABLE public.store TO " +
user.Id + ";\n"
    sentence6 += "GRANT UPDATE, SELECT ON TABLE public.user_info TO " +
 user.Id + ";\n"
    fmt.Println(sentence6)
    _, err = db.Query(sentence6)
    fmt.Println(" create a role successfully")
    //给用户随机发精灵，本来想用触发器写，但是因为仓库命名采用用户名+仓库拼接命名，
无法实现，所以单独写了
    rand.Seed(time.Now().UnixNano())
    num := rand.Intn(251) + 1
```

```
    var name string
    rows := db.QueryRow("select pokemon_name from pokemon where
pokemon_id = '" + strconv.Itoa(num) + "';")
    rows.Scan(&name)
    sentence8 := "insert into " + user.Id + "_warehouse values('" +
strconv.Itoa(num) + "','" + name + "', now());"
    _,err = db.Query(sentence8)
    //调用存储过程给新用户发福利
    sentence7 := "call welfare('"+ user.Id +"');"
    _,err = db.Query(sentence7)
    return err
  } else {
    fmt.Println("fail to create a role")
    return err
  }
}
```

- 自动补货功能：我们打算提供充足的精灵球，但是直接不设置库存或者设置库存为无限有些违背逻辑。因此我们设定了自动补货**函数**并通过**触发器**调用，在库存少于10的时候自动补货

  事实上，我们还写了指定补货数量的函数，可以由超级管理员用户调用

```
create function restock()
returns trigger as $$
begin
  if(NEW.stock<10) then
    update store set stock = stock+100 where item_id = NEW.item_id;
  end if;
  return NUll;
end; $$
language plpgsql;

CREATE TRIGGER auto_restock
AFTER UPDATE ON store
for each row
execute procedure restock();
```

- 修改用户信息

  指定新的用户信息并输入，根据主码找到这个用户，并且修改他的密码，权限等信息

- 删除用户

  指定用户id，先解除各个表对于用户信息的授权，在按照外码依赖的顺序删除用户仓库和用户背包，再删除用户。

```
func delete_role(Id string){
  str_table:= "REVOKE ALL ON TABLE public.store FROM " + Id + ";\n"
  str_table+= "REVOKE ALL ON TABLE public.skill FROM " + Id + ";\n"
  str_table+= "REVOKE ALL ON TABLE public.pokemon FROM " + Id + ";\n"
  str_table+= "REVOKE ALL ON TABLE public.pokemon_skill FROM " + Id +
";\n"
  str_table+= "REVOKE ALL ON TABLE public.user_info FROM " + Id + ";\n"
  str_table+= "drop table " + Id + "_backpack;\n"
  str_table+= "drop table " + Id + "_ballpack;\n"
  str_table+= "drop table " + Id + "_warehouse;\n"
  str_user:= "drop user " + Id + ";"
  str_table+=str_user
  _, err:=db.Query(str_table)
  checkErr(err)
  if err == nil {
    fmt.Println("delete successfully!")
  }else{
    fmt.Println("fail to delete a user")
  }
}
```

用户功能

- 注册

主要调用了管理员功能中提到的create_role函数，在注册之前先登录了管理员账号postgres，然后创建用户，然后退出postgres账号。

```
func Register_handler(c *gin.Context){
  var u Struct_login
  u.Id ="postgres"
  u.Password = "1908"
  var str string
  str = "user=" + u.Id + " password=" + u.Password + " dbname=DataBasePJ
sslmode=disable"
```

```
  db, err = sql.Open("postgres", str)
  //先登录超级管理员账号方便注册
  var user User_info
  c.ShouldBindJSON(&user)
  user.Id = "user_" + user.Id
  user.Password = encode_password(user.Password)//对密码加密
  fmt.Println(user)
  err2 := create_role(user)
  db.Close() //注册结束，退出超级管理员账号
  checkErr(err)
  if err2 == nil {
    c.String(http.StatusOK, "create a role successfully")
  } else {
    c.String(http.StatusNotAcceptable, "fail to create a role,%s", err2)
  }
}
```

我们为新注册的用户提供了福利，具体是5000块钱和一只随机的宝可梦。本来这两个都想通过触发器来写，但是由于我们用户个人仓库的命名方式是用户id+"_warehouse"的方式拼接建表命名的，因此无法在数据库内指明这个表，只能go内拼接好语句传入。所以触发器只做了送钱的部分，送宝可梦的部分丢进了go里实现

触发器如下：

```
create or replace procedure welfare(new_id varchar(20))
as $$ begin
  update user_info set user_money = user_money+5000 where user_id = new_id;
  commit;
end; $$
language plpgsql;
```

- 登录

用户登录时，会同时加载用户的个人信息，包括ID，昵称，宝可梦总数，金钱；

此外，还会加载用户背包里的精灵的所有信息，这些信息缓存在前端，方便战斗的时候快速加载和显示

```
func Login_handler(c *gin.Context) {
  c.ShouldBindJSON(&user)
  if user.Id!="postgres" && user.Id!= "ssyl" {
```

```
    user.Id = "user_" + user.Id
    user.Password = encode_password(user.Password)//对密码加密
  }
  db, err = login(user, "DataBasePJ")
  if err != nil {
    c.String(http.StatusNotAcceptable, "登录失败")
  } else {
    var poke_id [6]string
    i := 0
    rows3, err := db.Query("select pokemon_id from " + user.Id +
"_backpack;")
    for rows3.Next() {
      rows3.Scan(&poke_id[i])
      i++
    }
    fmt.Println(poke_id)
    checkErr(err)
    var result Result
    var poke_info Pokemon_info_withskill
    for j := 0; j < i; j++ {
      //pokmeon_details是一个返回宝可梦所有信息的函数
      poke_info.info, poke_info.skill = pokemon_details(poke_id[j])
      result.pokemons = append(result.pokemons, poke_info)
    }
    c.String(http.StatusOK, "[")
    fmt.Println(result)
    for j := 0; j < len(result.pokemons); j++ {
      c.JSON(http.StatusOK, gin.H{
        "id":              result.pokemons[j].info.Id,
        "name":            result.pokemons[j].info.Name,
        "property1":       result.pokemons[j].info.Property1,
        "property2":       result.pokemons[j].info.Property2,
        "attack":          result.pokemons[j].info.Attack,
        "special_attack":  result.pokemons[j].info.Special_Attack,
        "defense":         result.pokemons[j].info.Defense,
        "special_defense": result.pokemons[j].info.Special_Defense,
        "health":          result.pokemons[j].info.Health,
        "speed":           result.pokemons[j].info.Speed,

        "skill_name1":    result.pokemons[j].skill[0].Skill_name,
        "skill_type1":    result.pokemons[j].skill[0].Skill_type,
        "attack_type1":   result.pokemons[j].skill[0].Attack_type,
        "attack_power1":  result.pokemons[j].skill[0].Attack_power,
```

```
        "self_damage1":   result.pokemons[j].skill[0].Self_damage,
        "attribute1":     result.pokemons[j].skill[0].Attribute,
        "effect_id1":     result.pokemons[j].skill[0].Effect_id,
        "effect_result1": result.pokemons[j].skill[0].Effect_result,

        "skill_name2":    result.pokemons[j].skill[1].Skill_name,
        "skill_type2":    result.pokemons[j].skill[1].Skill_type,
        "attack_type2":   result.pokemons[j].skill[1].Attack_type,
        "attack_power2":  result.pokemons[j].skill[1].Attack_power,
        "self_damage2":   result.pokemons[j].skill[1].Self_damage,
        "attribute2":     result.pokemons[j].skill[1].Attribute,
        "effect_id2":     result.pokemons[j].skill[1].Effect_id,
        "effect_result2": result.pokemons[j].skill[1].Effect_result,

        "skill_name3":    result.pokemons[j].skill[2].Skill_name,
        "skill_type3":    result.pokemons[j].skill[2].Skill_type,
        "attack_type3":   result.pokemons[j].skill[2].Attack_type,
        "attack_power3":  result.pokemons[j].skill[2].Attack_power,
        "self_damage3":   result.pokemons[j].skill[2].Self_damage,
        "attribute3":     result.pokemons[j].skill[2].Attribute,
        "effect_id3":     result.pokemons[j].skill[2].Effect_id,
        "effect_result3": result.pokemons[j].skill[2].Effect_result,

        "skill_name4":    result.pokemons[j].skill[3].Skill_name,
        "skill_type4":    result.pokemons[j].skill[3].Skill_type,
        "attack_type4":   result.pokemons[j].skill[3].Attack_type,
        "attack_power4":  result.pokemons[j].skill[3].Attack_power,
        "self_damage4":   result.pokemons[j].skill[3].Self_damage,
        "attribute4":     result.pokemons[j].skill[3].Attribute,
        "effect_id4":     result.pokemons[j].skill[3].Effect_id,
        "effect_result4": result.pokemons[j].skill[3].Effect_result,
      })
      if j < 3 {
        c.String(http.StatusOK, ",\n")
      }
    }
    c.String(http.StatusOK, "]\n")
  }
}
//加载用户的个人信息，包括ID，昵称，宝可梦总数，金钱；
func User_info_handler(c *gin.Context){
  rows5, err := db.Query("select user_name, user_money from user_info where
user_id = '" + user.Id + "';")
```

```
  checkErr(err)
  var name string
  var money int
  for rows5.Next() {
    rows5.Scan(&name, &money)
  }
  rows2, err := db.Query("select count(*) from " + user.Id + "_warehouse;")
  checkErr(err)
  var num int
  for rows2.Next() {
    rows2.Scan(&num)
  }
  if err != nil {
    c.String(http.StatusNotAcceptable, "登录失败")
  } else {
    c.JSON(http.StatusOK, gin.H{
      "id":          user.Id,
      "name":        name,
      "pokemon_num": num,
      "money":       money,
    })
  }
}
```

此外再介绍一下pokmon_details函数，这个函数返回了宝可梦的所有信息，由于登录功能的使用是非常非常频繁的，所有我们在登录时的加载过程中运用了**视图**，来便于查询。

```
func pokemon_details(id string) (Pokemon_info, [4]Skill_info) {
  var info Pokemon_info
  var s1 string
  var s2 string
  var s3 string
  var s4 string
  rows, err := db.Query("select* from pokemon_detail where pokemon_id = '"
+ id + "';")
  //pokemon_details 是数据库的里的一个视图，具体定义见解释
  checkErr(err)
  for rows.Next() {
    err = rows.Scan(&info.Id, &info.Name, &info.Property1,
      &info.Attack, &info.Special_Attack, &info.Defense,
      &info.Special_Defense, &info.Health, &info.Speed, &info.Property2,
&s1, &s2, &s3, &s4)
```

```
  }
  var ret [4]Skill_info
  ret[0] = get_skills_details(s1)
  ret[1] = get_skills_details(s2)
  ret[2] = get_skills_details(s3)
  ret[3] = get_skills_details(s4)
  return info, ret
}

func get_skills_details(name string) Skill_info {
  rows, _ := db.Query("select* from skill where skill_name = '" + name +
"';")
  var s Skill_info
  s.Effect_id = make([]int, 5)
  s.Effect_result = make([]int, 5)
  for rows.Next() {
    rows.Scan(&s.Skill_name, &s.Skill_type, &s.Attack_type,
&s.Attack_power,
      &s.Self_damage, &s.Attribute, &s.Effect_id, &s.Effect_result)
  }
  return s
}
```

视图的定义如下

```
CREATE OR REPLACE VIEW public.pokemon_detail
 AS
 SELECT pokemon.pokemon_id,
    pokemon.pokemon_name,
    pokemon.property,
    pokemon.attack,
    pokemon.special_attack,
    pokemon.defense,
    pokemon.special_defense,
    pokemon.health,
    pokemon.speed,
    pokemon.sec_property,
    pokemon_skill.skill1,
    pokemon_skill.skill2,
    pokemon_skill.skill3,
    pokemon_skill.skill4
   FROM pokemon
```

```
     JOIN pokemon_skill USING (pokemon_id);

ALTER TABLE public.pokemon_detail
    OWNER TO postgres;
```

- 查看仓库里的宝可梦：

返回仓库里的宝可梦的信息，包括宝可梦名字，属性，抓到的时间

- 从仓库里将宝可梦加入背包

首先需要打开仓库，然后根据看到的信息，选择宝可梦加入到背包中。

- 从背包里取出宝可梦

将背包里的宝可梦删除

- 查看基本的用户信息

包括用户id，用户名称，宝可梦总数，用户金钱数

- 购买精灵球

向用户的背包加入精灵球，并且扣钱

由于在数据库中加入了断言：用户的金钱必须非负，因此如果用户金钱不够，将无法购买并返回错误。

```
func Buy_pokeball_handler(c *gin.Context) {
  var info Buy
  c.ShouldBindJSON(&info)
  str := "update user_info set user_money = user_money - " +
strconv.Itoa(info.Money) +
    " where user_id = '" + user.Id + "';"
  fmt.Println(str)
  _, err := db.Query(str)
 //事先加入了断言：用户的金钱必须非负，因此如果用户没钱了，购买就不会发生，返回状态码失
败。
  if err == nil {
    user_pack := user.Id + "_ballpack"
    str = "update " + user_pack + " set ball_num = ball_num + 1 where
ball_type = " + strconv.Itoa(info.Type) + ";"
    fmt.Println(str)
```

```
    db.Query(str)
    str2:="update store set stock = stock-1 where ball_type = " +
strconv.Itoa(info.Type) + ";"
    db.Query(str2)
    c.String(http.StatusOK, "购买成功")
  } else {
    c.String(http.StatusNotAcceptable, "无法购买！")
  }
}
```

- 放生宝可梦

在用户仓库里删去宝可梦，由于外码依赖的存在，只有不在背包里的宝可梦可以被放生

- 抓宝可梦

对战成功并且运气够好，用户抓到了宝可梦，会想向后端返回一个宝可梦id和时间，在用户仓库里插入宝可梦。

对战和抓宝可梦的逻辑在前端实现，于游戏中呈现。同时，用户还有机会获得奖励（金钱）。

- 战斗

用户将在地图上随机遇到野生宝可梦，并发生战斗。战斗过程中，有一套完整的伤害和技能的战斗逻辑。

- 退出登录

断开与数据库的连接，游戏客户端仍然保持打开。

- 关于加密

使用了sha256包进行加密，数据库中存储的是加密后的秘钥

```
func encode_password(code string) string{
   encode := sha256.Sum256([]byte(code))
   hashcode:=hex.EncodeToString(encode[:])
   return hashcode
}
```

五、特色和创新点

如果你的系统有任何功能或技术上的特色或创新之处，请在这里加以说明。

设计了一个可以玩的游戏：有生动可操作的场景，完善的人机对战逻辑

使用了为用户创建role，对于公共表向用户提供授权，为用户建立了归属于用户自己的仓库和背包，从而更好地进行用户的权限管理，而非单纯在用户信息中加条目来模拟“用户”的存在

使用了安全性更高的SHA256加密方式来确保用户信息不被破解

使用了饱受好评的21世纪高效编程语言：golang

六、实验分工

对于两人一组的同学，请**务必**在此处说明实验分工，而且越细越好。独立完成的同学不 用写分工。

黄绵秋同学主要负责前端的设计和实现，包括：设计了包括登录场景、战斗场景、室外场景和室内场景一共四个游戏场景，设计了游戏的操控方式和游戏模式，并设计了一套与之匹配的简单AI，并利用http协议与后端进行通信，利用返回的结果进行可视化处理。爬到了需要的数据，并对于后端编写中的一些问题给予建议和指导。

宋宇霖同学主要负责后端的设计和实现，包括数据库设计和建立，数据库函数编写，数据库信息导入，go连接数据库以及控制数据库完成各种操作（具体见代码部分），以及基于http协议搭建本地服务器及编写路由和提供接口。

两人一起参与的工作有：项目背景构建，实验报告撰写（各自写自己的部分），游戏实现细节的讨论，

七、提交文件说明

对提交的源代码，数据库脚本进行说明。

由于Mac os系统的postgreSQL脚本导出比较奇怪，会出一些BUG，尝试了好多方法都没有成功，所以最终提交了两份数据库备份文件，一份是文本格式的，里面包含了所有表的SQL定义语句，触发器，函数和过程的定义，以及数据库中的数据，方便助教老师查看。另一份是数据库备份文件，可以在pgadmin里用于还原。

提交的源代码包括：

1. 数据库备份文件，文件名为：DataBasePJ
2. 数据库备份文件（文本格式方便查看），文件名为DataBasePJ(文本)
3. go后端源代码，文件名为：DataBasePJ后端.zip，解压后可以用golang或者支持go的编译器打开
4. 前端代码：前端.zip
5. 包括资源集、JavaScript脚本代码、cocos引擎代码在内的游戏包，文件名为Pokemon.zip，因尚未部署，因而需要拥有cocos creator2.45才能进行运行，之后会将该部分进行部署。

八、实验总结

总结本次实验，你说学习到的内容，以及碰到的问题等。

这次实验真的困难重重，我们花了相当相当多的时间，熬了好多夜，奋战了贼久才搞出成果，最终的成果也不能让我们完全满意，可以说还有很大的提升空间。

分析原因，主要是我们两个人尝试了此前完全没有接触过的语言和框架：go语言和CocosCreater游戏引擎，再加上游戏制作的逻辑相对于图书管理更加复杂，传递的参数的类型更多更复杂。我们花了很多时间学习引擎的使用，学习语言的功能和各种包各种函数，也因为不熟悉不了解很多语言提供的便捷方法，在一些操作上绕了点路，并且有一些想要实现的功能因为没有找到合适的函数无法实现。此外，由于框架不是特别成熟，加上环境不是很稳定，游戏引擎-go-pgsql的连接有时候会出问题，我们遇到了很多很多的bug，找到这些bug的解决方案和修复花费了非常长的时间。但是在这个过程中，我们熟悉了GO语言的许多语法和特性，也对于数据库的各种操作（包括平时比较少碰到的用户权限管理，表的权限管理，架构和schema，以及多值属性的实现等）有了更深的理解，认识到了数据库应用的广泛性。

当然，采用这样的语言和框架也是我们深思熟虑过的。GO是很高效的语言，并且原生很好地支持并发，这对于一个游戏而言是非常重要的，因为多用户登录是游戏的必经之路，高效的处理也有利于游戏的流畅运行。此外，GO的包管理系统很优秀，可以很方便地调用想用的包。CocosCreater作为一个比较轻量的游戏引擎，相对易学也容易拿来做一个小项目。虽然限于课业压力和有限的时间，我们还没有把它部署到服务器上并实现玩家对战，但是从发展角度来看，这在未来给这个项目留下了很大的空间。可以说，相比于千篇一律的图书管理系

统，这是一个有实际意义和可玩性的作品，这也是为什么我们愿意投入很多时间来做它，并且收获了很多乐趣。