```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;

namespace CupPlaner.Helpers
{
    // Takes care of the schedule functions: generate and clear
    public class ScheduleManager
    {
        // tries to schedule every match in a tournament that is to be played on a field
with size = fSize.
        // it is also restricted to numberOfFields number of fields
        public bool scheduleAll(int tournamentID, FieldSize fSize, int numberOfFields)
        {
            CupDBContainer db = new CupDBContainer();
            MatchGeneration mg = new MatchGeneration();
            Validator validator = new Validator();
            Tournament t = db.TournamentSet.Find(tournamentID);
            //set op a list of every tournamentStage, a list of all group stage
tournamentStages and a list of all matches
            List<TournamentStage> TournamentStages = db.TournamentStageSet.Where(x =>
x.DivisionTournament.Division.Tournament.Id == t.Id &&
x.DivisionTournament.Division.FieldSize == fSize).ToList();
            List<TournamentStage> TournamentStagesToSchedule = TournamentStages.Where(x
=> !x.Pool.IsAuto).ToList();
            List<Match> allMatches = db.MatchSet.Where(x =>
x.TournamentStage.DivisionTournament.Division.Tournament.Id == t.Id &&
x.TournamentStage.DivisionTournament.Division.FieldSize == fSize).ToList();

            //selector is used to get the next tournamentStages in the list if the
previous one was not usable
            int selector = 0;
            //dayCount restricts the number of days available for the algorithm at the
start. it will be incremented as the algorithm goes on
            int dayCount = 1;
            //indicator is used to select either the first or the last match from a
tournamentStage
            int indicator = 1;
            //IsScheduled is used to see if the algorithm is successfull
            bool IsScheduled = false;
            while (!IsScheduled)
            {
                //list of all unscheduled tournamentStages ordered by number of matches,
in decending order
                List<TournamentStage> unscheduledTournamentstages =
TournamentStagesToSchedule.Where(x => !x.IsScheduled).OrderByDescending(x =>
x.Matches.Count(y => !y.IsScheduled)).ToList();

                //if there is no more unscheduled tournamentStages, we are either done
with the groupstages of done with the shole schedule
                if (unscheduledTournamentstages.Count == 0)
                {
                    if (TournamentStagesToSchedule.All(x => !x.Pool.IsAuto))
                    {
                        TournamentStagesToSchedule.Clear();
                        TournamentStagesToSchedule = TournamentStages.Where(x =>
x.Pool.IsAuto).ToList();
                        continue;
                    }
                    else
                    {
                        IsScheduled = true;
                        continue;
                    }
                }
                //if after an update to the number of unscheduled tournamentstages the
selector is out of range, reset it
                if (selector >= unscheduledTournamentstages.Count)
                {
                    selector = 0;
                }
                // select the first tournamentStage in the list unless it is not usable
use the next, if that is not usable select the next and so on.
                TournamentStage ts = unscheduledTournamentstages.ElementAt(selector);
                {
                    //check if any teams has a previous pool that is not scheduled yet
                    bool isReady = true;
                    foreach (Team team in ts.Pool.Teams)
                    {
                        if (team.PrevPool == null)
                        {
                            continue;
                        }
                        else if (team.PrevPool.TournamentStage.IsScheduled)
                        {
                            if (ts.TimeInterval.StartTime <
team.PrevPool.TournamentStage.TimeInterval.EndTime)
                            {
                                ts.TimeInterval.StartTime =
team.PrevPool.TournamentStage.TimeInterval.EndTime;
                            }
                        }
                        else
                        {
                            isReady = false;
                            break;
                        }
                    }
                    if (!isReady)
                    {
                        selector++;
                    }
                    else
                    {
                        //get all uncheduled matches in the tournamentStage
                        List<Match> unscheduledMatches = ts.Matches.Where(x =>
!x.IsScheduled).ToList();
                        Match matchToSchedule;
                        //set the tournamentStage to scheduled in there are no
unscheduled matches
                        if (unscheduledMatches.Count == 0)
                        {
                            DateTime lastMatchStart = ts.Matches.Max(x => x.StartTime);
                            ts.TimeInterval.EndTime =
lastMatchStart.AddMinutes(ts.DivisionTournament.Division.MatchDuration * 2);
                            ts.IsScheduled = true;
                            continue;
                        }
                        //select the first or last match
                        else if (indicator > 0)
                        {
                            matchToSchedule = unscheduledMatches.First();

                        }
                        else
                        {
                            matchToSchedule = unscheduledMatches.Last();
                            //if we get the last match increment selector. this is done
such that we go to the next tournamentStage if this match is not schedule in the
following code
                            selector++;
                        }

                        //list of all fields
                        List<Field> fields =
matchToSchedule.TournamentStage.DivisionTournament.Division.Tournament.Fields.Where(x =>
x.Size == fSize).Take(numberOfFields).ToList();
                        List<Field> fieldsNotChecked = new List<Field>();
                        fieldsNotChecked.AddRange(fields);
                        //goes through each day available so far
                        for (int i = 0; i < dayCount; i++)
                        {
                            //order the fields by number of matches on the particular day
                            fieldsNotChecked = fieldsNotChecked.OrderBy(x =>
x.Matches.Count(y => y.StartTime.Date ==
x.NextFreeTime.ElementAt(i).FreeTime.Date)).ToList();
                            //check is the match can be scheduled at any fields
nextFreeTime
                            foreach (Field field in fieldsNotChecked)
                            {

                                if (validator.areTeamsFree(matchToSchedule,
field.NextFreeTime.ElementAt(i).FreeTime))
                                {
                                    matchToSchedule.StartTime =
field.NextFreeTime.ElementAt(i).FreeTime;
                                    matchToSchedule.Field = field;
                                    field.NextFreeTime.ElementAt(i).FreeTime =
field.NextFreeTime.ElementAt(i).FreeTime.AddMinutes(matchToSchedule.Duration);
                                    matchToSchedule.IsScheduled = true;
                                    db.Entry(field).State =
System.Data.Entity.EntityState.Modified;
                                    break;
                                }

                            }

                            if (matchToSchedule.IsScheduled)
                            {
                                break;
                            }
                        }
                        // if the match is scheduled reset our selector and indicator
                        if (matchToSchedule.IsScheduled)
                        {
                            indicator = 1;
                            selector = 0;
                            db.Entry(matchToSchedule).State =
System.Data.Entity.EntityState.Modified;
                        }
                    }
                }
                // if we got through every tournamentStage and all of the where not
usable, this will force a match into the schedule by adding minutes to the fields
nextFreeTimes untill a mach will fit
                if (selector >= unscheduledTournamentstages.Count )
                {
                    List<Match> allUnscheduledMatches = allMatches.Where(x =>
!x.IsScheduled).ToList();
                    //newDayCount to force a match in at the first day possible
                    int newDayCount = 1;
                    // viriable to increase the nextFreeTimes for the fields
                    int k = 0;
                    //shile loop that keeps going untill a new match is scheduled or the
algorithm fails
                    bool done = false;
                    List<Field> fields =
allUnscheduledMatches.First().TournamentStage.DivisionTournament.Division.Tournament.Fiel
ds.Where(x => x.Size == fSize).Take(numberOfFields).ToList();
                    while (!done)
                    {
                        k += 10;
                        //if every fields nextFreeTime added k minutes is passed the
endtime of the tournament each give this part of the algorithm an extra day, give the
shole algorithm an extra day or
                        // return false
                        if (fields.All(x => x.NextFreeTime.ElementAt(newDayCount -
1).FreeTime.AddMinutes(k) > t.TimeIntervals.ElementAt(newDayCount - 1).EndTime))
                        {
                            if (newDayCount < dayCount)
                            {
                                newDayCount++;
                                k = 0;
                                continue;
                            }
                            else if (dayCount < t.TimeIntervals.Count())
                            {
                                dayCount++;
                                done = true;
                                continue;
                            }
                            else
                            {
                                return false;
                            }
                        }
                        // go through each match and see if it can be scheduled at any
field with the added k minutes
                        foreach (Match match in allUnscheduledMatches.Where(x =>
x.TournamentStage.TimeInterval.StartTime != DateTime.MinValue))
                        {

                            List<Field> fieldsNotChecked = new List<Field>();
                            fieldsNotChecked.AddRange(fields);
                            fieldsNotChecked = fieldsNotChecked.OrderBy(x =>
x.Matches.Count(y => y.StartTime.Date == x.NextFreeTime.ElementAt(newDayCount -
1).FreeTime.Date)).ToList();
                            foreach (Field field in fieldsNotChecked)
                            {
                                if (field.NextFreeTime.ElementAt(newDayCount -
1).FreeTime.AddMinutes(k) >= t.TimeIntervals.ElementAt(newDayCount - 1).EndTime)
                                {
                                    continue;
                                }
                                if (validator.areTeamsFree(match,
field.NextFreeTime.ElementAt(newDayCount - 1).FreeTime.AddMinutes(k)))
                                {
                                    field.NextFreeTime.ElementAt(newDayCount -
1).FreeTime = field.NextFreeTime.ElementAt(newDayCount - 1).FreeTime.AddMinutes(k);
                                    match.StartTime =
field.NextFreeTime.ElementAt(newDayCount - 1).FreeTime;
                                    match.Field = field;
                                    field.NextFreeTime.ElementAt(newDayCount -
1).FreeTime = field.NextFreeTime.ElementAt(newDayCount -
1).FreeTime.AddMinutes(match.Duration);
                                    match.IsScheduled = true;
                                    db.Entry(field).State =
System.Data.Entity.EntityState.Modified;
                                    break;
                                }
                            }
                            if (match.IsScheduled)
                            {
                                selector = 0;
                                done = true;
                                break;
                            }
                        }
                    }
                }
                // get the indicator to the other side of 0 so we get the other end of
the tournamentStages
                indicator *= -1;
            }
            db.SaveChanges();
            return true;
        }

        //calculates the minimum number of fields needed for every single match to be
scheduled within the timeIntervals of a tournament
        public int MinNumOfFields(int tournamentID, FieldSize fs)
        {
            CupDBContainer db = new CupDBContainer();
            Tournament t = db.TournamentSet.Find(tournamentID);
            int duration = 0;
            foreach (Division d in t.Divisions)
            {
                if (d.FieldSize == fs)
                {
                    foreach (TournamentStage ts in d.DivisionTournament.TournamentStage)
                    {
                        duration += (d.MatchDuration * ts.Matches.Count());
                    }
                }

            }
            double tDuration = 0;
            foreach (TimeInterval ti in t.TimeIntervals)
            {
                tDuration += (ti.EndTime - ti.StartTime).TotalMinutes;
            }
            return (int)Math.Ceiling((duration / tDuration));
        }

        // Deletes the whole schedule for a tournament
        public void DeleteSchedule(int tournamentID)
        {
            CupDBContainer db = new CupDBContainer();
            DeleteSchedule(tournamentID, db);
        }
        public void DeleteSchedule(int tournamentID, CupDBContainer db)
        {
            //CupDBContainer db = new CupDBContainer();
            MatchGeneration mg = new MatchGeneration();
            Tournament t = db.TournamentSet.Find(tournamentID);
            if(db.MatchSet.Any(x =>
x.TournamentStage.DivisionTournament.Division.Tournament.Id == tournamentID))
            {
                foreach (Division d in t.Divisions.ToList())
                {
                    // Remove all division tournaments and their dependencies
                    if (d.DivisionTournament != null)
                    {
                        foreach (TournamentStage ts in
d.DivisionTournament.TournamentStage.ToList())
                        {
                            foreach (Match m in ts.Matches.ToList())
                            {
                                foreach (Team team in m.Teams.ToList())
                                {
                                    team.Matches.Remove(m);
                                }
                            }
                            db.MatchSet.RemoveRange(ts.Matches);
                            db.TimeIntervalSet.Remove(ts.TimeInterval);
                        }

db.TournamentStageSet.RemoveRange(d.DivisionTournament.TournamentStage);
                        db.DivisionTournamentSet.Remove(d.DivisionTournament);
                    }
                    // Remeove each pool that is generated automatically by the match
generation class and their dependencies
```

```
                    foreach (Pool pool in d.Pools.ToList())
                    {
                        pool.FavoriteFields.Clear();
                        if (pool.IsAuto)
                        {
                            foreach (Team team in pool.Teams)
                            {
                                db.TimeIntervalSet.RemoveRange(team.TimeIntervals);
                            }
                            db.TeamSet.RemoveRange(pool.Teams);

                            db.PoolSet.Remove(pool);
                        }
                    }
                }
                // Reset next free time of each field to default (tournament start time)
for each day
                TimeInterval[] tournamentTi = t.TimeIntervals.ToArray();
                foreach (Field f in t.Fields)
                {
                    db.NextFreeTimeSet.RemoveRange(f.NextFreeTime);
                    for (int i = 0; i < tournamentTi.Count(); i++)
                    {
                        f.NextFreeTime.Add(new NextFreeTime() { FreeTime =
tournamentTi[i].StartTime });
                    }
                }
                t.IsScheduled = false;
                db.Entry(t).State = EntityState.Modified;
                db.SaveChanges();
            }
        }
    }
}
```